COCHCOMO: An extension of COCOMO II for Estimating Effort for Requirement Changes during Software Development Phase

SUFYAN BASRI, NAZRI KAMA, ROSLINA IBRAHIM
Advanced Informatics School
Universiti Teknologi Malaysia
Kuala Lumpur
MALAYSIA
msufyan4@live.utm.my, {mdnazri, iroslina.kl}@utm.my http://www.ais.utm.my

Abstract: - Software undergoes changes at all stages of the software development process. Accepting too many changes will cause expense and delay and rejecting the changes may cause customer dissatisfaction. One of the inputs that help the software project management to decide whether to accept or reject the changes is by having a reliable estimation of the change effort. From a software development perspective, the estimation has to take into account the inconsistent states of software artifacts across project lifecycle i.e., fully developed or partially developed. This inconsistent state requires different ways of estimation such as the fully developed artifacts may have different calculation compared to the partially developed artifacts. Many change effort estimation models have been developed and one of them is using impact analysis. One main challenge of this technique from software development perspective is that this technique is specifically used for software maintenance phase in which all software artifacts have been completely developed. This research introduces a new change effort estimation model that is able to use different estimation techniques for different states of software artifacts. The outcome of this research is a new change effort estimation model for software development phase using the extended version of the static and dynamic analysis techniques from our previous works. The significant achievements of the approach are demonstrated through an extensive experimental validation using several case scenarios.

Key-Words: - Software development, change impact analysis, change effort estimation, impact analysis, effort estimation

1 Introduction

It is important to manage the changes in the software to meet the evolving needs of the customer and hence, satisfy them [1-3]. Accepting too many changes causes delay in the completion and it incurs additional cost. Rejecting the changes may cause dissatisfaction to the customers. Thus, it is important for the software project manager to make effective decisions when managing the changes during software development. One type of information that helps to make the decision is the estimation of the change effort produced by the changes. This prediction can be done by combining two most related concepts which are impact analysis and effort estimation.

On one hand, impact analysis is a process of identifying potential consequences of change, or estimating what needs to be modified to accomplish a change [4-5]. The motivation behind the impact analysis activity is to identify software artifacts (i.e., requirement, design, class and test artifacts) that are potentially to be affected by a change. On the other hand, change effort estimation is the process of predicting how much work and how many hours of work are needed for a particular change request. In recent project management processes, the effort invested in a project has become one of the most significant and most studied subjects.

Challenge with the current change effort estimation approaches [6-10] that uses impact analysis technique as the source of input is that there is no consideration of the inconsistency states of software artifacts across the project. This consideration is crucial since in the software development phase: (1) some artifacts are partially developed and; (2) some of them have been developed conceptually but not technically (or have yet been implemented). The failure of this consideration will lead to inaccuracy of estimation that eventually contributes to project delay or customer dissatisfaction.

This paper extends our previous works on change impact analysis approach [11-13] to support change effort estimation during software development phase. In brief, the previous approach integrates between the static analysis and dynamic analysis techniques to perform change impact analysis during the software development phase. In this paper, we extend the current approach's capability to support change effort estimation by introducing a new change effort estimation process. Due to space limitation, this paper will give more explanation on the new change effort estimation process rather than the previously developed change impact analysis approach. However, detail information on the change impact analysis approach can be referred to [11-13].

This paper is laid out as follow. Section 2 presents related work, whereas Section 3 introduces a new approach for change effort estimation. Section 4 explains our evaluation procedure and its results. Finally, Section 5 describes conclusion and future works.

2 Related Work

This section presents two most related keywords with our research which are impact analysis and effort estimation.

2.1 Impact Analysis

There are two categories of impact analysis techniques [12, 13] which are the static analysis technique and the dynamic analysis technique. The static analysis technique develops a set of potential impacted classes by analyzing program static information that is generated from software artifacts (i.e., requirement, design, class and test artifacts). Conversely, for the dynamic analysis technique, this technique develops a set of potential impacted classes by analyzing program dynamic information or executing code.

Static Analysis: There are two most related current static analysis techniques to the new proposed approach which are the Use Case Maps (UCM) technique [14] and the class interactions prediction with impact prediction filters (CIP-IPF) technique [15, 16]. The UCM technique [14] performs impact analysis on the functional requirements and the high level design model. This technique assumes that all the functional requirements and the high level design model are completely developed. This technique has two limitations which are: (1) there being no traceability technique used from the functional requirements and the high level design artifacts to the actual source code. This technique only makes an assumption that the content of these two artifacts that is represented using the UCM model are reflected to the class artifacts. Any affected elements in the UCM model are indirectly reflected to the affected class artifacts; and (2) there is no dynamic analysis or source code analysis involved in this technique. Based on the precept that some of the effect of a change from a class to other class(es) may only be visible through dynamic or behavior analysis of the changed class [17, 18], results from this technique tend to miss some actual impacted classes.

Next, the CIP-IPF technique [15, 16] uses a class interactions prediction as a model for detecting impacted classes. This technique has its strength compared to the UCM technique. Comparing to the UCM technique, this technique has traceability link detection between the requirements artifacts and the class artifacts feature. This feature is used to navigate impact of changes at the requirement level to the class artifacts.

Dynamic Analysis: For the dynamic analysis techniques, we have selected two most related works to our research which are the Influence Mechanism technique [17] and the Path Impact technique [18]. Essentially, these techniques predict the impact set (classes or methods) based on method level analysis.

The Influence Mechanism technique [17] introduces the Influence Graph (IG) as a model to identify impacted classes. This technique uses the class artifacts as a source of analysis and assumes that the class artifacts are completely developed. There is a limitation of this technique which is there is no formal mapping or traceability process from requirements artifacts or design artifacts to class artifacts. This process is important in the impact analysis process as changes not only come from class artifacts, but it also comes from design and/or requirements artifacts. Since the design and requirements artifacts do interact among them vertically (between two different artifacts of a same type) and horizontally (between requirement and design artifacts), changes that happen to them could contribute to different affected class artifacts. In some circumstances, focusing on the source code analysis may not able to detect those affected classes.

The Path Impact technique [18] uses the Whole Path DAG (Directed Acyclic Graph) model as a model to identify impacted classes. The concept of implementation of this technique is almost similar to the Influence Mechanism technique as this

technique uses the class artifacts as a source of analysis and assumes that the class artifacts are completely developed. Also, this technique performs a preliminary analysis prior to performing a detailed analysis. There are two limitations of this technique. First, the implementation is time consuming as the technique opens a huge number of data when the analysis goes to a large application. Next, there is no formal mapping process from requirements artifacts or design artifacts to class artifacts. As described earlier, this process is important in the impact analysis process as changes doesn't only come from class artifacts, but also from design and/or requirements artifacts.

2.2 Effort Estimation

There are several categories of effort estimation which are: (1) Expert Judgment [6]; (2) Estimation by Analogy [7]; (3) Function Point Analysis [8]; (4) Regression Analysis [9]; and (5) Model Based [10].
A study by Jorgensen [6] shows that, expert judgment in effort estimation is one of the most common approaches today. Now more project managers prefer to use this method instead of formal estimation models, while the other techniques are simply more complex and less flexible than expert judgment methods. There is currently no method in effort estimation, which can prove its result to be hundred percent accurate. So, project managers just prefer to accept the risks of estimation and perform the expert judgment method for their effort estimation.

Effort estimation by analogy uses information from the similar projects which has been developed formerly, to estimate the effort needed for the new project. The idea of analogy-based estimation is to estimate the effort of a specific project as a function of the known efforts from historical data on similar projects. This technique could be combined with machine learning approaches to automate and to become more effective [7].

Traditionally, software size and effort measured using LOC (Lines of Code) measure. However, earlier studies [8] show that when the scale of the development grows, estimating using LOC fails to achieve accurate software effort estimation. Also, using different languages could be a problem; different languages could create different values of LOC. The addressed problems could be solved by using Function Point in software measurement and estimation. Function Point Analysis uses Function Point (FP) as its measure; therefore, it is suggested for improving the software measurement and estimation methods.

Another way to estimate software development effort is to use regression analysis; also known as algorithmic estimation. It uses variables for software size such as LOC and FP as independent variables for regression-based estimation and mathematical methods for effort estimation [19, 20]. Some multiple regression models also use other parameters such as development programming language or operating system as extra independent variables. The advantage of regression models is their mathematical basis as well as accuracy measurements.

3 A New Change Effort Estimation Approach

As described earlier, the new change effort estimation is an extension of our previous work on change impact analysis for the software development phase. The new approach basically introduces a new effort estimation model that modifies the current COCOMO II technique [21].

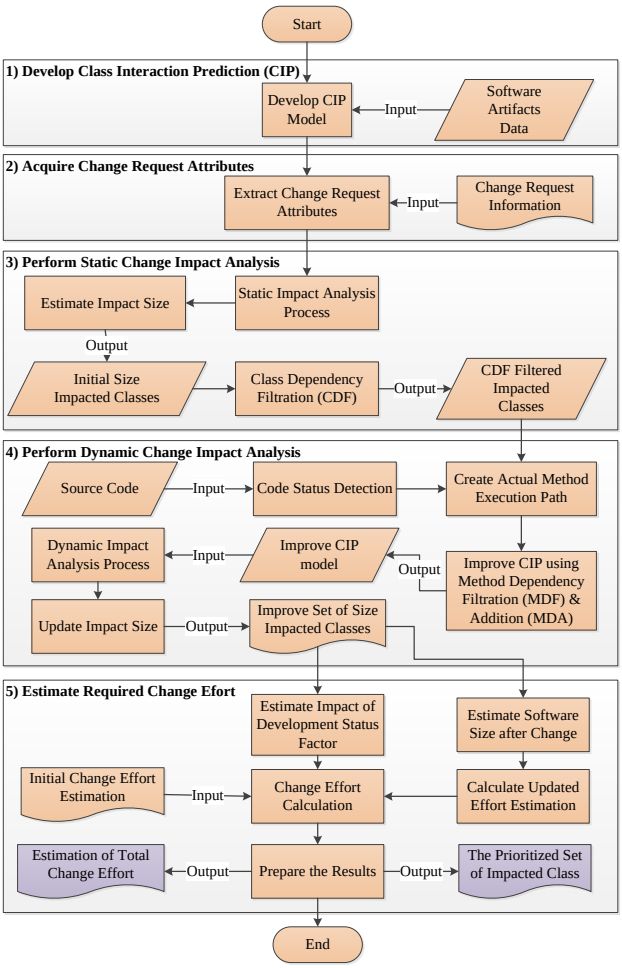

Fig. 1 Overall change effort estimation approach

The modified version which we called Constructive Change Cost Model (COCHCOMO) technique will use our previously developed change impact analysis approach.

Fig. 1 shows the overall change effort estimation process. Basically there are five steps which will be discussed in following sections of this document: (1) Developing class interactions prediction (CIP) model; (2) Acquiring change request attributes; (3) Performing static impact analysis; (4) Performing dynamic impact analysis; (5) Estimating required change effort.

3.1 Step 1: Developing Class Interactions Prediction Model

This step develops the Class Interaction Prediction (CIP) model based on inference of requirements interactions. The CIP model basically is a traceability model that shows the interactions between all the software artifacts including requirements, designs and classes. It will be used later to perform static change impact analysis and find the impacts of the change. Detail of the development of CIP model can be referred to [12, 13].

3.2 Step 2: Acquiring Change Request Attributes

The second step of COCHCOMO model is to acquire the change request attributes. This step intends to identify any relevant attributes of the change request which has direct impact on the effort estimation results. According to one of the previous works [22], among the relevant attribute is type of change.

3.3 Step 3: Performing Static Impact Analysis

This step performs static impact analysis on the developed CIP model to find the direct and indirect affected classes. The static impact analysis will find any direct impacted classes, which are the first layer of classes affected by a particular changed requirement without vertical traceability relations consideration. Then indirect impacted classes (second or next level of impacted classes) will be identified by completing traceability search through the CIP model.

This step uses a Breadth-first search (BFS) algorithm [23] to perform static impact analysis to search for the impacted classes in the CIP model. The algorithm states that software artifacts in CIP model are the nodes of the search graph and the impacted classes are the goals of search.

The process continues by performing a static filtration on the results, to remove some of falsely predicted results by over estimation. The method we use for static filtration is class dependency filtration (CDF). The CDF process finds the potential but falsely identified impacted classes by tracing the flow of interactions between the classes. If any indirect impact class could not be traced back to any of the direct impacted classes; that class is considered as falsely predicted impact class and it is filtered from the impact analysis results. It uses a cut-set from CIP model which contains the vertical relations between the classes to perform a backward tracing search from indirectly impacted classes to the directly impacted classes. The CDF filtering produces the final static impact analysis results that will be used by the dynamic impact analysis step. Detailed explanation of this step can be referred to [24].

This step also calculates an estimation of the potential impact size of each impacted class based on the CDF results. Potential impact size is number in percentage that will be used to predict the size of code after implementing the change. The Impact Size Factor (ISF) is calculated using the following equation [25]:

$$ISF_{IC} = 100 \times \sum_{r=0}^{n} PV_{AT} \frac{CTF_r}{NR} \quad (1)$$

where ISF_{IC} is impact size factor for impacted class (IC), r is relation from requirement to the impacted class based on the developed CIP model in step 1 due to the change. The relation refers to the interactions that exists derived from the traceability model among the software artifacts including requirements, designs and classes. When a change occurred and affected certain requirement(s), relationship from the affected requirement to the impacted class is established. NR is number of requirement artifacts that have relation to the impacted class, PV_{AT} is a constant value for probability of change for affect type (AT) – where AT could be *direct* or *indirect* affection type and value for $PV_{AT} < 1$. *Direct* or *indirect* affection type refers to the type of relationship that exists from the affected requirement to the impacted class. In our experiment, we have defined value 0.9 for *direct* and 0.7 for *indirect* affect type. CTF_r is change type factor based on the affected requirement change type which lead to the relation r (see Table 1).

ISF is the sum of probable occurrence of change in each identified impacted class according to the change type factor for that change request. Using eqs (1) we are able to have a prediction of ISF for each identified impacted class. Although, this value is just a rough estimation, and it is not very accurate, this equation can produce an overall good prediction of the future class size expansion after the change with calibrated variables.

CTF value is a coefficient of correlation value between -1 and 1. It is used to predict the effect of the change type on the impact size. The proposed change types and its value can be referred to Table 1 [25].

Change Type	Value	Effect
Addition	1.00	All of the code with estimated size will be added to the original code.
Modification-Major Grow	0.75	Three quarter of the estimated impact code will be added to the original code.
Modification-Grow	0.50	Only half of the estimated impact code will be added to the original code.
Modification-Minor Grow	0.25	One quarter of the estimated impact code will be added to the original code.
Modification-Negligible	0.10	The change is so insignificant that code size will not be changed much.
Modification-Minor Shrink	-0.25	One quarter of the estimated impact code will be removed from the original code.
Modification-Shrink	-0.50	Only half of the estimated impact code will be removed from the original code.
Modification-Major Shrink	-0.75	Three quarter of the estimated impact code will be removed from the original code.
Deletion	-1.00	All of the estimated impact code will be removed from the original code.

Table 1 Change Type Factor Values

3.4 Step 4: Performing Dynamic Impact Analysis

This step aims to find the actual interactions among classes by performing Method Dependency Filtration (MDF) analysis. Overestimated static impact analysis results that come from CDF Filtration are expected to be removed by MDF filtration.

The main challenge in this step is to detect fully developed classes. We propose a new mechanism to identify status of the code i.e., (1) code status could be not developed; (2) partly developed or; (3) fully developed. The first step is to detect the classes which are not developed. The classes without declaring in the code files are considered as not developed classes. But still if the class declaration is available in the code, a concrete method is needed to distinguish the fully developed from the partly developed classes. For distinguishing them, a special tag for classes and methods is recommended to be developed to keep its code status.

The structure of the special tag for the code status tag should be as follow: [Special comments mark + "<status>" + Code Status + "</status>"], where special comments mark in Visual C++ is three slashes "///", and Code Status could be different according the programming methodology. Some of the possible code statuses are "Not Developed", "Stubbed", "Faked", "Mocked", "Partly Developed", or "Fully Developed". The methods with any code status but "Fully Developed" are considered as partly developed methods. Additionally, the classes with any code status but "Fully Developed" or having a partly developed method are considered as partly developed classes. In case that code status tags were not available for the methods and development status of the class could not be determined by the code status tag, an additional procedure to detect partly developed classes is used. This additional procedure detects stubs, fake methods, and incomplete methods.

Stubbed Method: A stubbed method is a dummy procedure used for linking a program with a partly developed library. The purpose of stubbed methods is to prevent "undefined label" errors at link time when the actual code is not developed. Normally, a stubbed method throws a not implemented exception in its first line, to prevent raising errors by the compiler. Therefore, any methods which throw an exception in their first line are considered as stubs.

Faked Method: A faked method is a method, which appears to be a functional method without any errors, but in fact it only returns a single constant value without performing any procedure. If a method returns a single constant value in its first line, it is considered as a faked method.

Incomplete Method: If a method is not stub or fake, and still, it does not perform its functionality fully, it is considered as incomplete. The methods hierarchies from the designs are compared with their

actual call hierarchy in code. If they are not the same, this method is not completely developed yet.

After detecting the partly and fully developed classes, the method execution paths are created from fully developed classes. The actual interaction between the classes can be determined from the created method executions paths. Afterwards, the CIP model is updated with the actual class interactions. Finally, the method dependency filtration (MDF) process is performed similar to CDF process on the impacted classes to filter the overestimated impact analysis results.

The improved filtered set of impacted classes by this process is the final impact analysis result in method. This method implies that by having fully developed classes we can perform accurate impact analysis, even with inaccurate CIP model from the beginning.

3.5 Step 5: Estimating Required Change Effort

The last and most important step is to estimate the required change effort based on the initial effort estimation and change impact analysis results. To estimate the change effort based on COCOMO II effort estimation [26], we propose a mathematical equation to calculate change effort (CPM) according to the original estimated effort (PM) and updated effort estimation (PM′). CPM is the total effort need to implement the change; it is equal to priority multiplier multiplied by the deviation of estimated effort with new software size (PM′) and original estimated effort (PM) plus the extra effort needed to change the developed code as follows:

$$CPM = \left((PM' - PM) + abs\left[(PM' - PM) \times DSF \right] \right) \times PR \quad (2)$$

where DSF is the development status factor based on eqs (8), PM is the original estimated effort using COCOMO II in man per month, PM′ is the updated estimated effort after change using new software size in man per month and it is calculated using eqs (3) and PR is the priority multiplier which is determined by the effect of the change request priority and how much it will affect the change effort; this value should be selected according to the development methodology of the development group.

Eqs (3)-(5) below show how PM′ is calculated. This equation will be justified with the assumption that the cost factors [22] and the scale factors [22] will not change with the change request. Accordingly, the mathematical justification for producing this equation is as follows:

$$PM' = \frac{PM'}{PM} \times PM \quad (3)$$

$$PM' = \frac{A \times CSize^{B} \times \left(\prod_{i=1}^{n} EM_i\right)}{A \times Size^{B} \times \left(\prod_{i=1}^{n} EM_i\right)} \times PM \quad (4)$$

$$PM' = \left(\frac{CSize}{Size} \right)^{B} \times PM \quad (5)$$

where PM is the original estimated effort using COCOMO II in man per month, PM′ is the updated estimated effort with new software size in man per month, B is one of the exponent in COCOMO II derived from the COCOMO II's five Scale Drivers as shown in eqs (6), Size is the original estimation of code size, CSize is the estimated code size after implementing the change.

$$B = B_0 + B_1 \times \sum_{i=1}^{5} SF_i \quad (6)$$

where B_0 and B_1 are COCOMO II's constant variables, SF stands for scale factor, which will be derived from the COCOMO II's five scale factors.

Assuming that the initial effort estimation was done before the change request, the only unknown variable in eqs (7) is CSize. Exponent B, PM, and Size are the known variables which can be easily obtained from the initial effort estimation. CSize is equal to the original estimated size plus additional size from impacted classes. The size of fully developed impacted classes can be calculated in dynamic change impact analysis process, but the size of other impacted classes should be provided according to the initial effort estimation. CSize is calculated by the following equation:

$$CSize = Size + \sum_{IC} \left(Size_{IC} \times ISF_{IC} \right) \quad (7)$$

where Size is equal to initial estimation of software size, IC stands for impacted class, $Size_{IC}$ is the size of the impacted class IC, ISFIC is the impact size factor for the impacted class IC which is calculated using eqs (1) in impact analysis steps.

DSF in eqs (2) is the development status factor. This value indicates how much extra effort is needed to change the impacted developed classes. This value will specify that, if the impacted class is a fully developed class, it will need more effort to change it than a partly developed class, and

moreover changing a partly developed class needs more effort than a not developed class. By using DSF in our calculation we are generalizing the fact that the change effort will intensively increase as more classes are being fully developed, and implement changes in early stages of development is less costly [26]. DSF will be calculated using the following equation:

$$DSF = \frac{(ND \times NND)+(PD \times NPD)+(FD \times NFD)-NIC}{NIC} \quad (8)$$

where DSF stands for development status factor (DSF ≥ 0), ND is equal to affect multiplier for not developed classes (see Table 3), NND is the number of not developed impacted classes, PD is equal to affect multiplier for partly developed classes (see Table 3), NPD is the number of partly developed impacted classes, FD is equal to affect multiplier for fully developed classes (see Table 3), NFD is the number of fully developed impacted classes, NIC is the total number of impacted classes.

The multipliers ND, PD and FD multipliers should be selected according to the phase distribution of the software development methodology used for the project. They can have different values for each project or development team. Moreover, there has been a research on the phase distribution of the development effort [21] which could be used to estimate multiplier values.

Table 2 shows how much effort is needed in each phase of the project in our experiment which is using RUP methodology. Accordingly, a sample of ND, PD and FD multiplier values are created in Table 3.

Phase	Schedule	Effort
Inception	10%	5%
Elaboration	30%	20%
Construction	50%	65%
Transition	10%	10%

Table 2 Phase Distribution Weight in RUP [27]

Multipliers	Related Phases	Value
ND	Inception, Elaboration	0.25
PD	Inception, Elaboration and a quarter of Construction	0.4125
FD	Inception, Elaboration and Construction	0.9

Table 3 Estimated Values for the Multipliers

In this research, COCHCOMO is developed for Early Design sub-model of COCOMO II [21] which uses SLOC as the software size metric. Therefore, we use logical SLOC as the code size; however, this model can easily be adapted for other COCOMO II sub-models [21] and also use of Function Points as software size metric.

4 Evaluation

This section describes the process of evaluating our new approach. Firstly, the case scenario and the controlled experiments used for its evaluation are defined and described. Then, a set of evaluation metrics is used to compare the actual results and experiment results. Later, procedure of evaluating this approach against similar approaches is described. Finally, evaluation results are then demonstrated and discussed.

4.1 Case Scenario

To measure the accuracy of the approach, we have implemented the approach in a small project which implements an On Board Automobile. The project consists of 1365 SLOC. Four case scenarios (see Table 4) are constructed to create different development progress states. Each case scenario was created to represent different types of development progress states in the selected project.

Case scenario	Progress	States description
CS1	Analysis	Software design is finished, but none of the classes are developed yet
CS2	Coding	Software design is finished, and some partly developed classes exist
CS3	Testing	All the classes are developed, and some of them are fully developed
CS4	Deployment	All the classes are fully developed, and the development phase is finished.

Table 4 Case Scenarios

4.2 Change Request

Considering four case scenarios (CS) with different development progress statuses and the change types, twenty change requests have been selected, and the distribution of selected change requests is presented in Table 5.

Change Type (CT)	CS1	CS2	CS3	CS4
CT1- Addition	CR1	CR6	CR11	CR16
CT2- Modification-Grow	CR2	CR7	CR12	CR17
CT3- Modification-Negligible	CR3	CR8	CR13	CR18
CT4- Modification-Shrink	CR4	CR9	CR14	CR19
CT5- Deletion	CR5	CR10	CR15	CR20

Table 5 Change Request Distribution

4.3 Evaluation Metrics

For evaluating the accuracy of the approach, three effort estimation metrics have been used which are Magnitude of Relative Error (MRE) [28], Mean Magnitude of Relative Error (MMRE) [29], and Percentage of Prediction, PRED (.25) [30].

MRE: The MRE is a metric for the absolute estimation accuracy only [28]. It calculates the rate of the relative errors in both cases of over-estimation or under-estimation as shown in eqs (9).

$$MRE = abs\left| \frac{Actual\ \text{Re}sults - Estimated\ \text{Re}sults}{Actual\ \text{Re}sults} \right| \quad (9)$$

MMRE: The MMRE or the Mean Magnitude of Relative Error is the percentage of average of the MREs over an entire data set [29]. It is used for calculating the accuracy of an estimation technique using T number of tests as it is shown in eqs (10).

$$MMRE = \frac{100}{t}\sum_{i}^{t} MRE_i \quad (10)$$

The MRE metric will be calculated for each predicted impacted class from the change request experience to measure the accuracy of the change effort estimation in COCHCOMO approach. But the MMRE will be calculated for the whole case scenario, which contains twenty change requests and several impacted classes. The results of our approach are more accurate when the MMRE values are smaller.

Percentage of prediction, PRED (.25) is percentage of estimates that fall within 25 percent of the actual value [30]. Percentage of prediction definition is shown in eqs (11), where K is the number of estimations where MRE value is less or equal to x and n is the total number of estimations.

$$PRED(x) = \frac{\mathrm{K}}{\mathrm{n}} \quad (11)$$

4.4 Evaluation Procedure

There are three main steps in the evaluation which are:

- Estimating change effort results using the new approach;
- Implementing actual changes to get actual change effort;
- Measuring the accuracy of the results.

The actual change effort is determined by measuring the time that has been taken to implement the change. The recorded time in eqs (12) below is used.

$$APM = \left(\frac{CTDEV + TDEV}{C \times \frac{SCED}{100}} \right)^{\frac{1}{SE}} - PM \quad (12)$$

where APM is actual change effort; PM is initial effort estimation value; TDEV is the initial calculated time deviation; SCED, C, and SE values are same as initial effort estimation values; CTDEV is change time deviation in months.

To calculate the CTDEV, we assume that the months are 30 days and the project will be working on 8 hours per day. Hence, the time deviation is equal to the actual taken time per hours for change multiplied by 30 × 8.

4.5 Threats to Validity

We have considered three threats of external validity to the evaluation results, which are:

(1) Project based on RUP methodology: The results of the experiment may constraint to projects which based on RUP. More extensive experiments need to be conducted if a project is using different types of development methodology (i.e. waterfall, agile, etc).

(2) Sample change request size: There are only twenty change requests data available for the experiment. This sample size may not strongly strengthen the experiment results. However, we believe that these results may have significant basis to conduct for further works in this research

(3) The use of student's project as the experiment's subject: The experiment data was based on project data which conducted by a group of post-graduate students in Advanced Informatics School, Universiti Teknologi Malaysia. Although it is not a real data from the software industry, the considerations are made based on the accessibility and the completeness of the data. Firstly, the post-graduate students are Master level student, who some of them has

experience in the software industry prior to joining the Master programme. Secondly, the post-graduate students are required to follow thorough software development process similar to software industry in which requirement gathering process, change request and management, complete documentation process and software development and testing activities are occurred. Thirdly, the activities conducted within a controlled environment which emulate the real software development activities only, without other confounding factors such as politics, business constraints and other external factor which always occurred in real software projects but might not relevant to the experiment and objectives of this research. Due to these considerations, we strongly believe that the results obtained are sufficient as a basis to conduct further works in the real software industry afterwards.

5 Result and Discussion

To recap, the evaluation will be focusing on comparing results between the estimated change effort with the actual change effort. We have used the MMRE and Percentage of Prediction, PRED (.25) as the comparison metric.

According to [31] most effort estimation techniques having difficulty to produce accurate effort estimation results as they produced more than 30% MMRE value compared to the actual results. In other study [30], proposed an acceptable MMRE value (or error rate) for software effort estimation is 25%. This value shows that on average, the accuracy of the estimation is more than 75%. For our evaluation, we have used this guideline to assess the accuracy of our proposed approach by targeting the MMRE value (or acceptable error rate) should be less than 25%. We also have used PRED (.25) as the second evaluation metric to support the result produced by MMRE.

Since our proposed model is a change effort estimation model and not general effort estimation model, we assume that the change effort is slightly smaller than the overall effort needed for developing a software package. Therefore, a small miscalculation or an error will cause a large relative error in the estimations, so it has been expected to have moderate accuracy in the proposed change effort estimation model. Table 6 shows the MRE, MMRE and PRED (.25) of change requests in each case scenario.

CT	CS1	CS2	CS3	CS4	Average
CT1	0.18327	0.01625	0.02714	0.13138	0.089508
CT2	0.11850	0.16912	0.03794	0.20112	0.131670
CT3	0.29487	0.56085	0.56917	0.32800	0.438222
CT4	0.16109	0.20672	0.21317	0.01538	0.149091
CT5	0.00812	0.00518	0.01000	0.08656	0.027467
MMRE	15.32%	19.16%	17.15%	15.25%	16.72%
PRED(.25)	80%	80%	80%	80%	80%

Table 6 MRE, MMRE and PRED(.25) based on Change Types (CT) across Case Scenario (CS)

Due to space limitation, a quick look on the average MMRE value revealed that: (1) our proposed model has 16.72% relative error on average which is better than our expectation; (2) all MMRE values for the case scenarios is less than 20%; (3) In term of percentage of prediction, PRED (.25) revealed that COCHCOMO accuracy is 80% for all case scenarios; and (4) On average, change type 3 (CT3) which require very small impact contributes to overall COCHCOMO's inaccuracies. This preliminary analysis indicated that the proposed COCHCOMO change effort estimation model is acceptably accurate. However, the accuracy results need to be further investigated and analyzed.

6 Conclusion

An effective change acceptance decision is one of the crucial factors to a success of failure of a software project. A software project manager needs to have a justified decision whenever a software change is introduced. Extending change effort estimation to the change impact analysis process able to provide significant justification for the change acceptance decision made.

In this paper, we have developed a constructive change cost model (COCHCOMO) which calculates the change effort estimation required to implement the change. The approach has extended the change impact analysis framework for software development phase to estimate change effort. This novel approach has taken into account the inconsistent states of software artifacts in its estimation process, i.e. partially developed or not developed class(es).

The results of this paper are part of an ongoing research to overcome the challenges of change acceptance decisions for the requested changes in

the software development phase. For future works, we aim to conduct an intensive test to this approach by considering more change requests from different software projects with specific characteristics, i.e. type, size, complexity, etc. Also, we will extend this approach for estimating change cost based on the approach results.

Acknowledgements:
The authors would like to thank to Ministry of Education Malaysia (MoE) and Universiti Teknologi Malaysia (UTM) for their financial support under Vote No: 4L067.

References:

[1] S. L. Pfleeger and S. A. Bohner, A framework for Software Maintenance Metrics, *Proceedings of the International Conference on Software Maintenance*, 1990, pp. 320-327.

[2] K. H. Bennet and V.T. Rajlich, Software Maintenance and Evolution: A Roadmap, *Proceedings of the International Conference on the Future of Sofware Engineering*, 2000, pp. 75-87.

[3] G. Kotonya and I. Somerville, *Requirements Engineering: Processes and Techniques*, John Wiley & Sons Ltd, 1998.

[4] R. S. Arnold, S.A. Bohner, Impact analysis-Towards a framework for comparison, *Software Maintenance, 1993*, CSM-93, Proceedings., Conference on. 27-30 Sep 1993, pp. 292-301.

[5] G. Antoniol, G. Canfora, and G. Casazza, Information Retrieval Models for Recovering Traceability Links between Source Code and Documentation, *Proceedings of the International Conference on Software Maintenance*, 2000, pp. 40-44.

[6] M. Jørgensen, Practical guidelines for expert-judgment-based software effort estimation, *Software, IEEE. 22(3)*, 2005, pp. 57-63.

[7] J. Li, G. Ruhe, A. Al-Emran and M. M. Richter, A flexible method for software effort estimation by analogy, *Empirical Softw. Engg. 12(1)*, 2007, pp. 65-106.

[8] Z. Yinhuan, W. Beizhan, Z. Yilong and S. Liang, Estimation of software projects effort based on function point, *Computer Science & Education, 2009. ICCSE '09*, 4th International Conference on 25-28 July 2009, 2009, pp. 941-943.

[9] C. A. L. Garcia and C. M. Hirata, Integrating functional metrics, COCOMO II and earned value analysis for software projects using PMBoK, *Proceedings of the 2008 ACM symposium on Applied computing*, Fortaleza, Ceara, Brazil, 2008, pp. 820-825.

[10] I. Attarzadeh, A. Mehranzadeh and A. Barati, Proposing an Enhanced Artificial Neural Network Prediction Model to Improve the Accuracy in Software Effort Estimation, *Computational Intelligence, Communication Systems and Networks (CICSyN)*, 2012 Fourth International Conference on. 24-26 July 2012, pp. 167-172.

[11] N. Kama and F. Azli, A Change Impact Analysis Approach for the Software Development, *Proceedings of Asia-Pacific Software Engineering Conference, APSEC.*, vol. 1, 2012, pp. 583-592.

[12] N. Kama, Integrated change impact analysis approach for the software development phase, *International Journal of Software Engineering and its Applications.*, vol. 7, no. 2, Mar 2013, pp. 293-304.

[13] N. Kama, Change impact analysis for the software development phase: State-of-the-art, *International Journal of Software Engineering and its Applications.*, vol. 7, no. 2, Mar 2013, pp. 235-244.

[14] J. Hassine, J. Rilling, J. Hewitt and R. Dssouli, Change Impact Analysis for Requirement Evolution using Use Case Maps, *Proceedings of the 8th International Workshop on Principles of Software Evolution*, 2005, pp. 81–90.

[15] N. Kama and F. Azli, Requirement Level Impact Analysis with Impact Prediction Filter, *Proceeding of the 2012 International Conference on Software Technology and Engineering (ICSTE 2012)*, Phuket Thailand, 1-2, September 2012, ASME, 2012, pp. 459-464.

[16] N. Kama, T. French and M. Reynolds, Predicting Class Interactions from Requirement Interactions: Evaluating a New Filtration Approach, *Proceedings of the IASTED International Conference on Software Engineering*, 2010, pp. 109-116.

[17] B. Breech, M. Tegtmeyer and L. Pollock, Integrating Influence Mechanisms into Impact Analysis for Increased Precision, *Proceedings of the 22nd International Conference on Software Maintenance*, 2006, pp. 55-65.

[18] J. Law and G. Rothermal, Whole Program Path-Based Dynamic Impact Analysis, *Proceedings of the 25th International*

Conference on Software Engineering (ICSE 2003), 2003, pp. 308-318.

[19] G. R. Finnie, G. E. Wittig and J. M. Desharnais, A comparison of software effort estimation techniques: Using function points with neural networks, case-based reasoning and regression models, *Journal of Systems and Software*, vol. 39, no. 3, 1997, pp. 281-289.

[20] S. Grimstad and M. Jørgensen, A framework for the analysis of software cost estimation accuracy, *Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering*, Rio de Janeiro, Brazil, 2006, pp. 58-65.

[21] D. Yang, Y. Wan, Z. Tang, S. Wu, M. He and M. Li, COCOMO-U: An Extension of COCOMO II for Cost Estimation with Uncertainty, *Q. Wang, D. Pfahl, D. Raffo & P. Wernick (Eds.), Software Process Change*, Springer Berlin Heidelberg, 2006, Vol. 3966, pp. 132-141.

[22] N. Nurmuliani, D. Zowghi and S. P. Williams, Requirements Volatility and Its Impact on Change Effort: Evidence-based Research in Software Development Projects, *11th Australian Workshop on Requirements Engineering*, University of Adelaide SA, 2006.

[23] R. Zhou and E. A. Hansen, Breadth-first heuristic search, *Artificial Intelligence*, vol. 170, no. 45, 2006, pp. 385–408.

[24] N. Kama, T. French and M. Reynolds, Impact Analysis using Class Interaction Prediction Approach, *Proceedings of the 2010 conference on New Trends in Software Methodologies, Tools and Techniques: Proceedings of the 9th SoMeT_10*, Hamido Fujita (Ed.). IOS Press, Amsterdam, The Netherlands, The Netherlands, 2010, pp. 96-111.

[25] M. H. Asl and N. Kama, A Change Impact Size Estimation Approach during the Software Development, *Software Engineering Conference (ASWEC), 2013 22nd Australian*, 4-7 June 2013, pp. 68-77.

[26] B. Sharif, S. A. Khan and M. W. Bhatti, Measuring the Impact of Changing Requirements on Software Project Cost: An Empirical Investigation, *IJCSI International Journal of Computer Science Issues. Vol. 9*, no. 3, 2012, pp. 170-174.

[27] P. Kruchten, *The Rational Unified Process: An Introduction*, Addison-Wesley, 2004.

[28] M. Jøgensen and K. Molokken-Ostvold, Reasons for software effort estimation error: impact of respondent role, information collection approach, and data analysis method, *IEEE Transactions on Software Engineering*, vol. 30, no. 12, 2004, pp. 993–1007.

[29] V. Nguyen, B. Steece and B. Boehm, A constrained regression technique for cocomo calibration, *Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement* (ESEM '08). ACM, New York, NY, USA, 2008, pp. 213-222.

[30] S.-J. Huang, N.-H. Chiu and L.-W. Chen, Integration of the grey relational analysis with genetic algorithm for software effort estimation, *European Journal of Operational Research. 188(3)*, 2008, pp. 898-909.

[31] S. Basha and D. Ponnurangam, Analysis of Empirical Software Effort Estimation Models, *International Journal of Computer Science and Information Security (IJCSIS)*, Vol. 7, No. 3, 2010.