A Novel Effort Estimation Approach for Requirement Changes during Software Development Phase

Sufyan Basri, Nazri Kama and Roslina Ibrahim

Advanced Informatics School, Universiti Teknologi Malaysia, Malaysia
msufyan4@live.utm.my, {mdnazri, roslina.kl}@utm.my

Abstract

Managing software changes and achieving an effective change acceptance decision is one of the crucial factors in ensuring the software project success. Accepting too many changes will cause expense and delay and rejecting the changes may cause customer dissatisfaction. One of the inputs that help in achieving an effective change acceptance decision is by having reliable estimation on the change effort. From 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. These inconsistent states require different ways of estimation such as the fully developed artifacts may have different calculation compared to the partially developed artifacts. Many change effort estimation approaches 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 approach 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 approach for software development phase using the extended version of the static and dynamic analysis techniques from our previous works.

Keywords: Software development, change impact analysis, change effort estimation, impact analysis, effort estimation

1. Introduction

An effective change acceptance decision is one of the crucial factors to a success or failure of a software project. It is very important to meet and satisfy the customer’s evolving needs in managing the software changes [1-3]. Accepting too many requirement changes causes delay in the completion and it incurs additional cost. Rejecting the requirement changes may cause dissatisfaction to the customers. Thus, it is important for the software project manager to make effective change acceptance decisions when managing the requirement changes during software development. One type of information that helps to make the decision is the estimation of the change effort produced by the requirement 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.
Generally most of the current change effort estimation approaches [6-10] that uses impact analysis technique as the source of input is that there is no consideration on 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). Failure on 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 [11-14] 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. This paper will give more explanation and focus 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 [11-14]. We’ve also developed a prototype tool that implements this new approach which we’ve describe in another paper [15].

This paper is laid out as follow. Section 2 presents related work whereas Section 3 introduces a new change effort estimation approach for software development phase. Section 4 and Section 5 explains our evaluation procedure and its results. Finally, Section 6 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 [16] and the class interactions prediction with impact prediction filters (CIP-IPF) technique [17, 18]. The UCM technique [16] 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 is no traceability technique used from the functional requirements and the high level design artifacts to the actual source codes. 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 [19, 20], results from this technique tend to miss some actual impacted classes.

Next, the CIP-IPF technique [17, 18] 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 [19] and the Path Impact technique [20]. Basically, these techniques predict the impact set (classes or methods) based on method level analysis.

The Influence Mechanism technique [19] 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 for 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 impact analysis process as changes not only come from class artifacts but it also comes from design and/or requirements artifacts. Since 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 [20] uses the Whole Path DAG (Directed Acyclic Graph) model as a model to identify impacted classes. The concept of implementation for 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 detail analysis. There are two limitations of this technique. First, the implementation is time consuming as the technique opens to 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 impact analysis process as changes not 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].

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 for automation 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 [21, 22]. 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 approach is an extension of our previous work on change impact analysis for the software development phase. The new approach basically introduces a new change estimation model that modifies the current COCOMO II technique [23]. The new modified version will use our previously developed change impact analysis approach.

Figure 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 the approach 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 [24], among the relevant attribute is type of change.
1) Develop Class Interaction Prediction (CIP)

2) Acquire Change Request Attributes

3) Perform Static Change Impact Analysis

4) Perform Dynamic Change Impact Analysis

5) Estimate Required Change Effort

Figure 1. Effort Estimation for Requirement Changes during Software Development Phase
### 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 [25] 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 [26].

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:

\[
ISF_{IC} = 100 \times \sum_r PV_{AT} \frac{CTF}{NR}
\]

Where,

- \( ISF_{IC} \): Impact size factor for impacted class (IC),
- \( r \): Relation from requirement to the impacted class,
- \( NR \): Number of requirement artifacts that have relation to the impacted class,
- \( PV_{AT} \): A constant value for probability of change for affect type (AT) – where AT could be direct or indirect affection type),
- \( CTF_r \) is change type factor based on the affected requirement change type which leads 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 a 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 [27].
3.4. Step 4: Performing Dynamic Impact Analysis

This step aims to find the actual interactions among 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 codes 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 codes, 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 is a dummy procedure used for linking a program with a partly developed library. 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 procedures. Incomplete method is not a stub or fake, and still, it does not perform its functionality fully, it is considered as incomplete.

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

### Table 1. Change Type Factor Values

<table>
<thead>
<tr>
<th>Change Type</th>
<th>Value</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>Addition</td>
<td>1.00</td>
<td>All of the code with estimated size will be added to the original code.</td>
</tr>
<tr>
<td>Modification-Major Grow</td>
<td>0.75</td>
<td>Three quarter of the estimated impact code will be added to the original code.</td>
</tr>
<tr>
<td>Modification-Grow</td>
<td>0.50</td>
<td>Only half of the estimated impact code will be added to the original code.</td>
</tr>
<tr>
<td>Modification-Minor Grow</td>
<td>0.25</td>
<td>One quarter of the estimated impact code will be added to the original code.</td>
</tr>
<tr>
<td>Modification-Negligible</td>
<td>0.10</td>
<td>The change is so insignificant that code size will not be changed much.</td>
</tr>
<tr>
<td>Modification-Minor Shrink</td>
<td>-0.25</td>
<td>One quarter of the estimated impact code will be removed from the original code.</td>
</tr>
<tr>
<td>Modification-Shrink</td>
<td>-0.50</td>
<td>Only half of the estimated impact code will be removed from the original code.</td>
</tr>
<tr>
<td>Modification-Major Shrink</td>
<td>-0.75</td>
<td>Three quarter of the estimated impact code will be removed from the original code.</td>
</tr>
<tr>
<td>Deletion</td>
<td>-1.00</td>
<td>All of the estimated impact code will be removed from the original code.</td>
</tr>
</tbody>
</table>
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 [28], 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 for change the developed code as the follow:

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

Where,

- DSF is the development status factor based on Equation (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 Equation (3),
- 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.

Equation (3)-(5) below shows how PM' is calculated. This equation will be justified with the assumption that the cost factors [24] and the scale factors [24] will not change with the change request. Accordingly, the mathematical justification for producing this equation is as follow:

$$ PM' = \frac{PM'}{PM} \times PM $$

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

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

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 the exponent derived from the five Scale Drivers using Equations (6)
• Size is the original estimation of code size,
• CSize is the estimated code size after implementing the change.

\[ B = B_o + B_1 \times \sum_{i=1}^{5} SF_i \]  \hspace{1cm} (6)

Where,
• B₀ and B₁ are constant variables;
• SF stands for scale factor, which will be derived from the five scale factors.

Assuming that the initial effort estimation was done before the change request, the only unknown variable in Equation (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:

\[ CSize = Size + \sum_{iC} \left( Size_{iC} \times ISF_{iC} \right) \]  \hspace{1cm} (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,
• ISF\_iC is the impact size factor for the impacted class\_iC which is calculated using Equation (1) in impact analysis steps.

DSF in Equation (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 [28]. DSF will be calculated using the following equation:

\[ DSF = \left( \frac{(ND \times NND) + (PD \times NPD) + (FD \times NFD) - NIC}{NIC} \right) \]  \hspace{1cm} (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 [23] which could be used to estimate multiplier values.

<table>
<thead>
<tr>
<th>Phase</th>
<th>Schedule</th>
<th>Effort</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inception</td>
<td>10%</td>
<td>5%</td>
</tr>
<tr>
<td>Elaboration</td>
<td>30%</td>
<td>20%</td>
</tr>
<tr>
<td>Construction</td>
<td>50%</td>
<td>65%</td>
</tr>
<tr>
<td>Transition</td>
<td>10%</td>
<td>10%</td>
</tr>
</tbody>
</table>

**Table 2. Phase Distribution Weight in RUP [29]**

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

In this research, the new approach is developed for Early Design sub-model of COCOMO II [23] which uses SLOC as the software size metric. Therefore, we use logical SLOC as the code size; however, this approach can easily be adapted for other COCOMO II sub-models [23] 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 study 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 Study

To measure the accuracy of the approach, we have implemented the approach in four case studies (see Table 4) that have different development progress states. Each case study was created to represent different types of development progress states in the software development phase.

<table>
<thead>
<tr>
<th>Case study</th>
<th>Progress</th>
<th>States description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CS1</td>
<td>Analysis</td>
<td>Software design is finished, but none of the classes are developed yet</td>
</tr>
<tr>
<td>CS2</td>
<td>Coding</td>
<td>Software design is finished, and some partly developed classes exist</td>
</tr>
<tr>
<td>CS3</td>
<td>Testing</td>
<td>All the classes are developed, and some of them are fully developed</td>
</tr>
<tr>
<td>CS4</td>
<td>Deployment</td>
<td>All the classes are fully developed, and the development phase is finished.</td>
</tr>
</tbody>
</table>

Table 4. Case Studies

4.2. Change Request

Considering four case studies (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.

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

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) [30], Mean Magnitude of Relative Error (MMRE) [31], and Percentage of Prediction, PRED (.25) [32].

MRE: The MRE is a metric for the absolute estimation accuracy only [30]. It calculates the rate of the relative errors in both cases of over-estimation or under-estimation as shown in (9).

\[
MRE = \frac{Actual\ Results - Estimated\ Results}{Actual\ Results} \times 100
\]

MMRE: The MMRE or the Mean Magnitude of Relative Error is the percentage of average of the MREs over an entire data set [31]. It is used for calculating the accuracy of an estimation technique using T number of tests as it is shown in (10).

\[
MMRE = \frac{100}{t} \sum MRE_i
\]
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 the approach. But the MMRE will be calculated for the whole case study, 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 [32]. Percentage of prediction definition is shown in (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.

\[
PRED(x) = \frac{K}{n}
\]  

(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 (12) below is used.

\[
APM = \left( \frac{CTDEV + TDEV}{C \times \frac{SCED}{100}} \right)^{\frac{i}{i_{x}}} - PM
\]  

(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 \( \times \) 8.

5. Results and Discussions

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 [33] 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 [32], 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 approach 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 approach. Table 6 shows the MRE, MMRE and PRED (.25) of change requests in each case study.
Table 6. MRE, MMRE and PRED(.25) Based On Change Types (CT) Across Case Study (CS)

<table>
<thead>
<tr>
<th>CT</th>
<th>CS1</th>
<th>CS2</th>
<th>CS3</th>
<th>CS4</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>CT1</td>
<td>0.18327</td>
<td>0.01625</td>
<td>0.02714</td>
<td>0.13138</td>
<td>0.089508</td>
</tr>
<tr>
<td>CT2</td>
<td>0.11850</td>
<td>0.16912</td>
<td>0.03794</td>
<td>0.20112</td>
<td>0.131670</td>
</tr>
<tr>
<td>CT3</td>
<td>0.29487</td>
<td>0.56085</td>
<td>0.56917</td>
<td>0.32800</td>
<td>0.438222</td>
</tr>
<tr>
<td>CT4</td>
<td>0.16109</td>
<td>0.20672</td>
<td>0.21317</td>
<td>0.01538</td>
<td>0.149091</td>
</tr>
<tr>
<td>CT5</td>
<td>0.00812</td>
<td>0.00518</td>
<td>0.01000</td>
<td>0.08656</td>
<td>0.027467</td>
</tr>
</tbody>
</table>

The average MMRE value revealed that: (1) our proposed approach has 16.72% relative error on average which is better than our expectation; (2) all MMRE values for the case studies is less than 20%; (3) In term of percentage of prediction, PRED (.25) revealed that our new approach’s accuracy is 80% for all case studies; and (4) On average, change type 3 (CT3) which require very small impact contributes to overall approach’s inaccuracies. This preliminary analysis indicated that our proposed effort estimation approach for requirement changes during software development phase is acceptably accurate. However, the accuracy results need to be further investigated and analyzed.

6. Conclusion

Software changes are inevitable in any software projects. It may occur at any point in Software Development Life Cycle (SDLC). Managing software changes and achieving an effective change acceptance decision is one of the crucial factors in ensuring the software project success. Software project manager has to balance between the need to fulfill the evolving user’s requirement and satisfaction as well as the need to meet project’s timeline and budget. Both require concrete justification and justified change acceptance decision especially during software development phase, there are a number of dynamic and inconsistent states of the software artifacts.

In this paper, we have introduced a novel change effort estimation approach for software development phase 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. In future, we aim to conduct more comprehensive experiments which cover more change requests from multiple projects with different characteristics i.e., type of development methodology, size, etc. We will also extend this approach for estimating change cost estimation using the result’s outcome.

Acknowledgements

The authors would like to thank to Ministry of Education Malaysia (MoE) and Universiti Teknologi Malaysia (UTM) for their financial support under Prototype Research Grant Scheme: 4L067.
References


Authors

Sufyan Basri, obtained his first degree at Universiti Teknologi Malaysia (UTM) in Electrical Engineering (Mechatronic) in 2001, second degree in Real Time Software Engineering at the same university in 2003. He is currently pursuing his PhD at UTM and his research interest includes Software Engineering, Change Management and Effort Estimation.

Nazri Kama, obtained his first degree at Universiti Teknologi Malaysia (UTM) in Management Information System in 2000, second degree in Real Time Software Engineering at the same university in 2002 and his PhD at The University of Western Australia (UWA) in Software Engineering in 2010. He has a considerable experience in a wide range on Software Engineering area. His major involvement is in software development.

Roslina Ibrahim, is currently attached to Advanced Informatics School (AIS) at Universiti Teknologi Malaysia, Kuala Lumpur. She holds a PhD in information science from Universiti Kebangsaan Malaysia and master degree in computer science. Her research interests are user acceptance of information systems, design and developments of educational games and usability evaluation. She has 15 years experiences in teaching several IT related courses both for undergraduate and postgraduate students, including IT and
multimedia, Web development, audio and video in multimedia, instructional multimedia design, IT Project Management, User Experience and Interaction Design and Research methodology. Currently, she is an Acting Academic Manager for Research Students with responsibilities of managing academic matters for Doctor of Philosophy and Master of Philosophy students at AIS.