Abstract: - Change effort estimation and impact analysis are two important methods to make effective change acceptance decisions for the software development phase. Accepting too many changes causes additional cost and delay in the competition. On the other hand, rejection of the changes may cause customer dissatisfaction. There are very few works have been done to support effective change decisions in development phase compared to the software maintenance phase. The challenge of estimating the change effort for the software development phase is because of the existence of partially developed artifacts. In this paper, we describe our initial model towards developing a new change effort estimation model for the software development phase.

Key-Words: - Software development phase, change impact analysis, change effort estimation, requirements interactions, requirements, class

1 Introduction

The decision of accepting or rejecting change requests has always been a complex process. This decision would be more uncertain during the software development, because of the large number of variables under dynamically changing requirements. This caused many software project managers to fail to reach a solid decision on applying changes during the software development phase.

Effort estimation and impact analysis are two of the important processes to support the decisions of software project managers. Impact analysis is the process of detecting the consequences or the effects of the requested changes on the software codes and documentations; and software efforts estimation is the process of predicting the most accurate amount of effort required to develop or maintain software based on a large number of changing variables. Later, the results from change impact analysis will be used as one of change effort estimation inputs to determine the change costs in terms of effort and schedule.

The challenges with the current effort estimation techniques are that first their results have to be revised each time a change in the requirements happens; and second accuracy of change impact analysis has direct effect on the accuracy of change effort estimation, while useful change effort estimation is performed using products of impact analysis. However, current impact analysis techniques don’t provide effort estimation with all the information it needs. In other words, lack of integration between change impact analysis and change effort estimation may decrease the usefulness and effectiveness of their results in the software development phase.

Accordingly, some researchers aimed to overcome the impact analysis and effort estimation challenges by improving each of these techniques separately [1, 2]. Now in this research, we identify the most practical impact analysis and change effort estimation techniques for the software development phase; then integrate them into a unified change impact analysis and effort estimation model. We believe, by this association, change effort estimation process would be both easier and more effective; which will help the software project managers and the change control board (CCB) to make change approval decisions with more confidence, and it will also reduce the overall cost of software development.

The Structure of this paper is as follow. First, we have a discussion about the background studies of this research which are: change management, impact analysis, and effort estimation (Section 2). Second, we introduce our change effort estimation model and its stages. Our model integrates SPD-CIF impact analysis technique by Kama et al. [3] with COCOMO II effort estimation technique by Boehm.
et al.[4] (Section 3). Then, we describe our initial evaluation process and its results (Section 4). Finally, we describe our conclusions and future works (Section 5).

2 Backgrounds
This section will describe two most related areas which are impact analysis and effort estimation.

2.1 Impact Analysis
Impact analysis is used to estimate the consequences of change requests as an origin for precise resource planning and scheduling, and to confirm the cost and effort validation. It determines the impacts of a change on the software and its related documentation if that proposed change is accepted. There are two approaches of impact analysis, which are dependency analysis and traceability analysis [3, 6].

Dependency analysis also known as program analysis is the analysis of relations only between source codes by exploring the organization of the codes. Traceability analysis uses the relationships between software artifacts for analysis; including requirements, design artifacts, source codes, and test artifacts; across different software phases.

Based on the impact analysis approaches, there are also two categories of impact analysis techniques accordingly. The impact analysis techniques are static analysis technique which are established by traceability analysis; and dynamic analysis technique which are established by dependency analysis [3, 6]. There is also another new type of impact analysis technique which is a combination of these two categories [3, 7].

2.1.1 Static Impact Analysis
The static impact analysis technique analyzes static information from software artifacts such as requirement, design, class, and test, to create a set of potential impacted classes [8, 9]. To perform static impact analysis technique, two categories of static information can be used which are high level artifacts and low level artifact. The low level artifact model uses source code model by reverse engineering from the existing source code or class artifacts to identify the set of potential impacted classes. On the other hand, high level artifact model uses the design and requirement artifacts to identify the set of potential impacted classes.

2.1.2 Dynamic Impact Analysis
The dynamic analysis technique analyzes dynamic information created by executing the code to create a set of potential impacted classes [8, 9]. In dynamic analysis, the dynamic information is generated by parsing through the source code. There are two categories of dynamic information to perform dynamic analysis which are the class dependency graph [10]; and the method execution paths [11].

2.1.3 Impact Analysis for Development Phase
However, studies [3, 7] show that combining static analysis and dynamic analysis techniques into one integrated technique [7]; and with considering partly developed classes would be a better approach for impact analysis software development phase [3]. Consequently, a framework has been developed for software development phase, which combines static and dynamic analysis; considers partly developed classes; and use all requirement, design, class artifacts for impact analysis. This framework [3] is called Software Development Phase Change Impact Analysis Framework (SDP-CIAF).

2.1.4 Organization of the SDP-CIAF
The SDP-CIAF [3] implementation has two main stages which are: Developing Class Interactions Prediction (CIP), and Performing Impact Analysis. In first stage the focus is to develop a CIP model [8] using requirement and design artifact; and in stage two the potential impacted classes will be identified from the CIP model and using filtration techniques on the results.

2.2 Effort Estimation
Effort estimation is the process of predicting how much work and how many hours of work are needed for a project to be completed. In recent project management processes, the effort invested in a project has become one of the most highlighted and most studied subjects.

The effort will affect the schedule and the cost of a project directly, which are two of the most important constraints in project management. As a result, this makes effort estimation process, a very critical activity in a software project; which plays an important role in the software project success or failure.

Realistic effort estimation could lead to effective plan for the cost and the schedule; while unrealistic effort estimation will cause over-estimation or under-estimation, which will consequently lead to software project failure. The estimation of the value of effort variable in project initiation allows us to
plan any approaching activities sufficiently. However, as changes appear, it is certain that effort estimation need to be revised each time a change is going to happen in the software project. The process of updating the effort estimation according to the change is called change effort estimation.

There are several methods for effort estimation. Researches have categorized the methods for effort estimation into several categories [12]. Here we briefly describe three of now most widely studied effort estimation techniques based on Jorgensen and Shepperd research [12] and later we will identify the most suitable method for change effort estimation to integrate with impact analysis technique for software development.

Correspondingly, there are also several approaches for implementing these effort estimation techniques; moreover, these techniques and approaches may also be combined with each other to create more accurate method.

2.2.1 Expert Judgment
The expert judgment effort estimation [13] is performed based on the human judgment instead of using models. It uses the latest discoveries of expert-based judgment on effort estimation which makes it flexible and very easy to implement.

2.2.2 Estimation by Analogy
Effort estimation [14] 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. Moreover, estimation by analogy may be classified as machine learning methods and composite methods.

2.2.3 Regression Analysis
The regression analysis; also known as algorithmic estimation; uses regression-based estimation and mathematical methods for effort estimation. One of the most commonly regression analysis techniques are Constructive Cost Model (COCOMO) II introduced by Boehm in 1995 [15]; yet it is an interesting effort estimation method which is a subject to some of recent researches [16, 17].

2.3 COCOMO II
The COCOMO II model [15] is the successor of the original COCOMO developed by Boehm in 1981 to address its constrains; it aims to develop a software cost and schedule estimation model for continuous estimation enhancements on whole software life cycle costs and schedules. One of the recommended approaches for measuring project size in COCOMO II is using Function Points[17].

However, the common metric for measuring the project size is KSLOC which stands for kilo source lines of code. SLOC is a logical line of code which is developed by programmers and is part of deliverable product. Filtering the partly developed classes in the previous stage will eliminate the possible errors in counting fake and test classes as actual developed code in this process. At this point, the accuracy of KSLOC is depended on how much of code has been developed; while this approach is for development phase.

3 A Change Effort Estimation Model
We propose that instead of two independent processes of impact analysis and change effort estimation, we can use an alternative approach. Our new model is combination of impact analysis and change effort estimation as a new approach; and it is called Change Acceptance Analysis (CAA). The results from CAA are going to help the CCB or project managers to decide accepting or rejecting changes by analyzing its consequences. In order to have effective CAA, the impact analysis technique must have the following features:

- It has to analyze all the software artifacts developed until development phase as well as the source code.
- It must include dynamic analysis on the code.
- It shall consider the partly developed classes; and exclude them from analysis.

We decided to use SDP-CIAF in our approach, while is has all the mentioned features. Also for change effort estimation, we needed a regression-based technique, because it could reduce the human error during this process; and also it can be recalculated with less effort and cost. Hence, we decided to use COCOMO II technique for change effort estimation; however, the original COCOMO II technique is for estimating the effort in the first phase of the project; so, this technique needed to be adapted for change effort estimation. In change effort estimation we should consider that effort was estimated at least once before; and it is possible to use the configuration management databases to get the initial change effort variables to compare with current products and change impacts.

Our CAA approach has three main stages, which are Developing Class Interactions Prediction, Performing Impact Analysis, and Estimating
Change Effort. Figure 2 shows the overall CAA approach; also here we give a brief description of these three stages in the following sections.

3.1 Stage 1: Developing CIP
The first stage is to develop the Class interactions Prediction (CIP) model. The CIP model contains the vertical and horizontal relationships between requirements, design artifacts, and classes. Vertical relationships are referred to the relations between same kind of artifacts; and horizontal relationships are referred to the relations between one type of artifact to the other types. This CIP model will be used to perform impact analysis in the next stage.

This stage has four processes which are: “Extracting Software Artifact Elements Process”, “Detecting Traceability Links”, “Developing Initial Class Interactions Prediction”, and “Modifying the Initial Class Interactions Prediction”. All these four processes would be performed sequentially; and the results from each of them, will provide the input for the next one. In the end, the final class interactions prediction model will be developed by the last process.

3.2 Stage 2: Performing Impact Analysis
Impact analysis will be performed on the developed CIP model from the previous stage. In this stage both static and dynamic analysis will be performed to create the final improved set of filtered impacted classes.

Hence, the second stage only has two main processes which are “Impact Analysis Process” and “Filtration Process”. The Filtration Process will remove falsely predicted results in two levels. The filtration levels are Class Dependency Filtration (CDF) which performs static analysis on the identified initial impacted classes; and Method Dependency Filtration (MDF) which performs dynamic analysis on the method execution paths according to CDF level results. In this framework, if a method proved to be a stub or partly developed, the MDF level will not be performed on that specific method. In the end, the improved set of filtered impacted classes developed by this framework proved to be an accurate impact analysis result for the software development phase.

During the process of performing impact analysis it should be considered that the details of change request must contain all the necessary information about the type and origin of the change as Sommerville [5] recommended. The change type will help to estimate the change effort in the next stage.

3.3 Stage 3: Estimating Change Effort
In this stage, the effort and time needed to change the impacted classes and artifacts will be estimated based on the improved set of filtered impacted classes, which was developed in stage 2. In this stage we use COCOMO II technique for effort estimation; though other techniques may be possible.
The first step for estimating the change effort is to identify which effort estimation variables need to be updated for that particular change request. In COCOMO II there are three types of variables that might change by the change request which are Software Size, Cost Drivers, and Scale Drivers.

After the new values for effort estimation variables have been determined, it is possible to continue to the next process which is estimating the change effort using the new values for its variables. This process may use the configuration management database to get the previous effort estimation results in order to speed up the process. The outcome of this process is the estimated change effort in person per month.

At last change effort schedule time deviation will be estimated using the previous stage results. In this process, the time needed to implement the requested change with consideration of effort and impacted classes will be estimated. There are two final results at the end of the process, which are the final prioritizes set of impacted classes which are prioritized by the effort needed to change each class; and total change effort and duration estimation. These two outputs are the final output of the CAA approach, which are enough to realize the consequences of change on the software project with respect of cost.

4 EVALUATION
There are two sub-sections in this section as follow:

4.1 Case Study Design
To evaluate the proposed CAA approach results, we plan to conduct an experiment using a sample case study data. The data we are using in this case study are collected from a small and easy to understand software development project, which has been developed by Rational Unified Process (RUP) methodology and it is developed by expert participants. The chosen participants are master of software engineering students, whom have experience in software development industry. Therefore, we can count them as expert participants. All the data related to changes and their impacts are gathered from this project as the actual results. The actual results from the gathered data and the results of this experiment will be compared for the purpose of evaluation.

First, we collect the actual data about the development process; change request; initial estimated effort; and actual time and effort they have put on the case study. Then we use their software development library to collect a version of source codes and documents in their development phase, where a change request has occurred. We use these artifacts to perform CAA on and then compare its results with the actual change impacts and efforts.

4.2 Evaluation Results
Here is a summary of the process we have accomplished to find the evaluation results. The first stage is to develop CIP model; first we extracted all software artifact elements; the software artifacts in this case study are requirement artifacts, design artifacts, and class artifacts. Then we identified the horizontal and vertical traceability links between them to create the initial CIP. Next step is to modify the initial CIP according to the design pattern to create the final CIP model.

Then we performed the impact analysis process according to the change request to identify the initial impacted classes. We excluded some of the classes during the filtration process to finally create the final set of improved filtered classes and percentage of their impacted methods.

The next stage is to estimate the change effort using updated effort estimation variables in the following equation:

\[
P_M = A \times \text{Size}^B \times \left( \prod_{i=1}^{n} EM_i \right)\]  

(1)

Where, \( PM \) is the estimated effort in person per month; \( A \) is equal to multiplicative constant; \( \text{Size} \) stands for software size (KSOCL); \( B \) is the exponent derived from the five Scale Drivers (see Table 1); and \( EM \) stands for effort multiplier which is derived from the seventeen Cost Drivers (see Table 2).

Table 1. COCOMO II Scale Factors

<table>
<thead>
<tr>
<th>No</th>
<th>Scale Factor</th>
<th>Sample Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>PREC</td>
<td>Nominal</td>
<td>Precedence of the project</td>
</tr>
<tr>
<td>2</td>
<td>FLEX</td>
<td>Low</td>
<td>Development flexibility</td>
</tr>
<tr>
<td>3</td>
<td>RESL</td>
<td>Nominal</td>
<td>Architecture and risk resolution</td>
</tr>
<tr>
<td>4</td>
<td>TEAM</td>
<td>Very High</td>
<td>Team interconnection</td>
</tr>
<tr>
<td>5</td>
<td>PMAT</td>
<td>Nominal</td>
<td>Process maturity, determined by SEI</td>
</tr>
</tbody>
</table>
Table 2. COCOMO II Cost Drivers

<table>
<thead>
<tr>
<th>No</th>
<th>Cost Driver</th>
<th>Sample Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>RELY</td>
<td>Nominal</td>
<td>Required Software Reliability</td>
</tr>
<tr>
<td>2</td>
<td>DATA</td>
<td>Low</td>
<td>Size of Database used (DB bytes/Pgm)</td>
</tr>
<tr>
<td>3</td>
<td>CPLX</td>
<td>Nominal</td>
<td>Complexity of the product</td>
</tr>
<tr>
<td>4</td>
<td>RUSE</td>
<td>Nominal</td>
<td>Required reusability</td>
</tr>
<tr>
<td>5</td>
<td>DOCU</td>
<td>Very High</td>
<td>Documentation match to life-cycle needs</td>
</tr>
<tr>
<td>6</td>
<td>TIME</td>
<td>High</td>
<td>Execution time constraint</td>
</tr>
<tr>
<td>7</td>
<td>STOR</td>
<td>Nominal</td>
<td>Main storage constraint</td>
</tr>
<tr>
<td>8</td>
<td>PVOL</td>
<td>Nominal</td>
<td>Platform volatility</td>
</tr>
<tr>
<td>9</td>
<td>ACAP</td>
<td>Low</td>
<td>Analyst capability</td>
</tr>
<tr>
<td>10</td>
<td>PCAP</td>
<td>Nominal</td>
<td>Programmer capability</td>
</tr>
<tr>
<td>11</td>
<td>PCON</td>
<td>Nominal</td>
<td>Personnel continuity</td>
</tr>
<tr>
<td>12</td>
<td>AEXP</td>
<td>Nominal</td>
<td>Application experience</td>
</tr>
<tr>
<td>13</td>
<td>PEXP</td>
<td>Nominal</td>
<td>Platform experience</td>
</tr>
<tr>
<td>14</td>
<td>LTEX</td>
<td>Nominal</td>
<td>Language and tools experience</td>
</tr>
<tr>
<td>15</td>
<td>TOOL</td>
<td>Nominal</td>
<td>Use of software tools</td>
</tr>
<tr>
<td>16</td>
<td>SITE</td>
<td>Nominal</td>
<td>Multisite development</td>
</tr>
<tr>
<td>17</td>
<td>SCED</td>
<td>Nominal</td>
<td>Scheduling factor</td>
</tr>
</tbody>
</table>

The new source codes needed to be added to these current initial source codes determined before the change are considered as “add” and its multiplicative constant is 1. If the new determined source codes already existed in initial source codes, but they need to be modified; it is considered as “modify” and its multiplicative constant is 0.5. At last the source codes that they no longer needed to be implemented are considered as “cancel” and its multiplicative constant is -1, while it will reduce the effort needed for implementing the whole product.

\[
\begin{align*}
\text{PM}_{\text{modify}} &= 0.5 \times 2.439^{1.01} \times 1.1521 = 1.4176 \\
\text{PM}_{\text{cancel}} &= -1 \times 0.378^{1.01} \times 1.1521 = -0.4313 \\
\text{PM}_{\text{change}} &= 0.7816 + 1.4176 - 0.4313 = 1.7679
\end{align*}
\]

Moreover, the duration of the project based on the estimated effort can be predicted by the effort time deviation with the equation below:

\[
TDEV = C \times PM^{SE} \times \frac{SCED}{100}
\]  \hspace{1cm} (2)

Where, \(TDEV\) is the calendar time in months; \(C\) is a constant which is selected according to the software project; \(SCED\) is nominal according to Table 2; and \(SE\) is the schedule equation which is calculated by the following equation: \(SE = D + 0.2 \times (E - B)\). Where, \(D\) is a constant; \(E\) stands for the effort scaling exponent derived from the five Scale Drivers (see Table 1); and \(B\) is calibrated scale factor base-exponent. Therefore, considering the effort needed to implement the change is 7.9172 + 1.7679 = 9.6851(p/m). The \(TDEV\) after the change is calculated as follow:

\[
TDEV = 3.67 \times (9.6851)^{0.33+0.2 \times (1.1521-1.11)} = 3.6011 \text{ months}
\]

Table 3 shows the results and the analysis done on the correctness of the proposed approach

<table>
<thead>
<tr>
<th>Model</th>
<th>CAA results</th>
<th>Actual Results</th>
<th>Correctness</th>
</tr>
</thead>
<tbody>
<tr>
<td>Impacted Classes</td>
<td>7 classes</td>
<td>7 classes</td>
<td>100%</td>
</tr>
<tr>
<td>Total Effort</td>
<td>9.6851 p/m</td>
<td>10.5 p/m</td>
<td>92%</td>
</tr>
<tr>
<td>Total Duration</td>
<td>3.6011 months</td>
<td>3.5 months</td>
<td>97%</td>
</tr>
</tbody>
</table>
5 CONCLUSIONS
This paper represents a new change effort estimation model that will be used to analyze a change request; and further, it helps the software project manager the make an acceptance decision in the software development phase. Our new model is called the Change Acceptance Analysis (CAA). The uniqueness of this model is that this model is meant to support change effort estimation for the software development phase. The support is done by extending our previous work on change impact analysis approach for the software development phase to change effort estimation. The first evaluation process proved this technique’s correctness; however, further improvement through evaluation on several software projects is needed to measure its effectiveness.

ACKNOWLEDGMENTS
The authors would like to thank Lab of Advanced Informatics School for the helps offered, and all members of Lab for their useful discussions that guide for this research. The financial of this project is supported by Ministry of Higher Education Malaysia and Universiti Teknologi Malaysia under Vot No: 00K01.

References: