New Trends in Software Methodologies, Tools and Techniques
H. Fujita et al. (Eds.)
IOS Press, 2014

doi:10.3233/978-1-61499-434-3-1029

COCHCOMO: A Change Effort Estimation Tool for Software Development Phase

Nazri KAMA[1], Sufyan BASRI[2], Mehran Halimi ASL[3] and Roslina IBRAHIM[4]
Advanced Informatics School, Universiti Teknologi Malaysia, MALAYSIA

Abstract. It is important for software project manager to make effective decisions when managing software 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. One of the inputs that help to decide whether to accept or reject the changes 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 and partially developed. This research introduces a new change effort estimation tool (Constructive Change Cost Model or COCHCOMO) that is able to take into account the inconsistent states of software artifacts in its estimation process. This tool was developed based on our extended version of static and dynamic impact analysis techniques. Extensive experiments using several case studies have been conducted in which the results show acceptable error rates have been achieved.

Keywords. Change effort estimation, change impact analysis, effort estimation, impact analysis, software development phase

Introduction

It is important to manage the changes in the software to meet the evolving needs of the customer and hence, satisfy them [1-5]. 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 [6-8]. The motivation behind the impact analysis activity is to identify software artifacts (i.e.,

[1] Corresponding Author: Nazri Kama, Advanced Informatics School, Universiti Teknologi Malaysia, Malaysia; E-mail: nazrikama@ic.utm.my
[2] Corresponding Author: Sufyan Basri, Advanced Informatics School, Universiti Teknologi Malaysia, Malaysia; E-mail: msufyan4@live.utm.my
[3] Corresponding Author: Mehran Halimi Asl, Advanced Informatics School, Universiti Teknologi Malaysia, Malaysia, E-mail: mehran.net1@gmail.com
[4] Corresponding Author: Roslina Ibrahim, Advanced Informatics School, Universiti Teknologi Malaysia, Malaysia, E-mail: lina@ic.utm.my

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 [9] 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.

We have extended our previous works on change impact analysis approach [9, 10] to support the development of change effort estimation model for the software development phase [11]. This paper focusing on the demonstration of the change effort estimation tool that has been built based on the previously developed change effort estimation model [11]. Also, we demonstrate the accuracy of the developed tool to support change effort estimation in the software development phase.

This paper is laid out as follow. Section 1 presents related work whereas Section 2 demonstrates our new change effort estimation tool. Section 3 evaluates the accuracy of the tool and followed by Section 4 discusses the results. Finally, Section 5 describes conclusion and future works.

1. Related Work

There are several categories of effort estimation which are: (1) Expert Judgment [12]; (2) Estimation by Analogy [13]; (3) Function Point Analysis [14]; (4) Regression Analysis [15, 16]; and (5) Model Based [17].

Study by Jorgensen [12] 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 [13].

Traditionally, software size and effort measured using LOC (Lines of Code) measure. However, earlier studies [14] 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 [18, 19]. 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.

2. A New Change Effort Estimation Tool

Our previously developed change effort estimation model [11, 20] has five main steps which are: (1) Developing class interactions prediction (CIP) model; (2) Acquiring change request attributes; (3) Performing static change impact analysis; (4) Performing dynamic change impact analysis; and (5) Estimating required change effort using change impact analysis results

Our new change effort estimation tool (which we called as COCHCOMO-Constructive Change Cost Model) has five main steps which are:

- Step 1: Importing Class Interactions Prediction (CIP)
- Step 2: Acquiring Change Request (CR)
- Step 3: Performing Impact Analysis
- Step 4: Estimating Change Effort
- Step 5: Analyze Results

Figure 1 shows the Main form of the tool that reflects to the five main steps of the model. We will describe in details each steps in the subsequent sections.

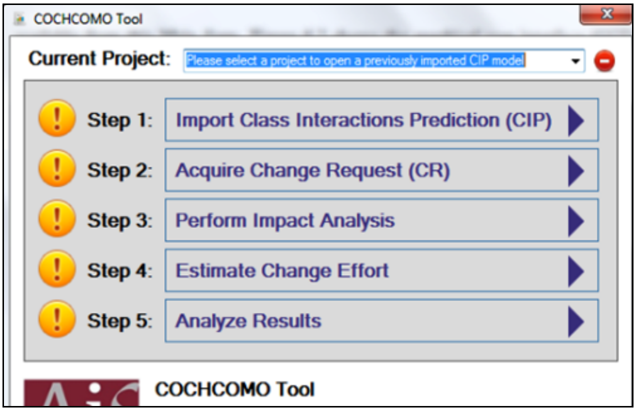

Figure 1. Main form user interface

2.1. Step 1: Importing Class Interactions Prediction (CIP)

In the first step, this tool requires the Class Interaction Prediction (CIP) [21] file to be imported in the database. There are two sub-steps in the import process which are extracting information from the CIP file, and saving the CIP information into the database. Later, the tool will allow the user to print the CIP report accordingly. The screen shot of the Import CIP form is shown in Figure 2:

Figure 2. Import CIP form

2.2. Step 2: Acquiring Change Request (CR)

Next, this step will get change request details from the user. We have designed a specific form based on the important change request attributes. The tool will automatically assign a unique identification number for each change request. Figure 3 shows the change request form.

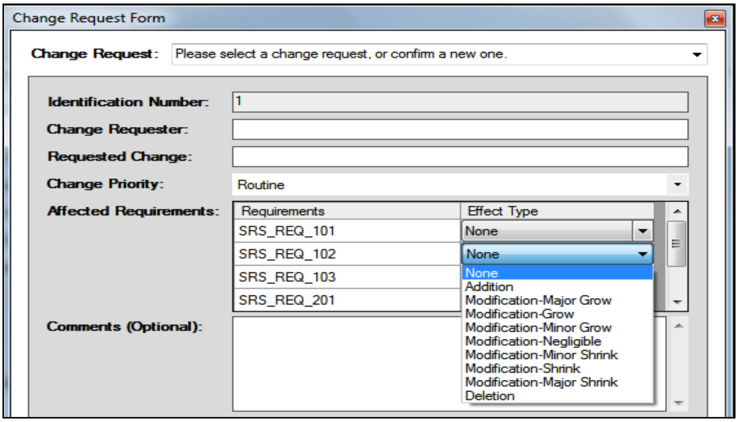

Figure 3. Change request form

2.3. Step 3: Performing Impact Analysis

There are three sub-steps to perform impact analysis: (1) perform static change impact analysis; (2) filter static impact analysis results using class dependency filtration and (3) perform dynamic change impact analysis. In performing static change impact analysis, we employ two levels of search. First, an optimized breadth-first search (BFS) algorithm [22] to be performed on the directed graph created from the CIP model without any vertical interactions to find the directly impacted classes. Next, a complete

BFS search will be performed on the overall CIP model to find all the indirectly impacted classes. Figure 4 shows the implementation of static change impact analysis.

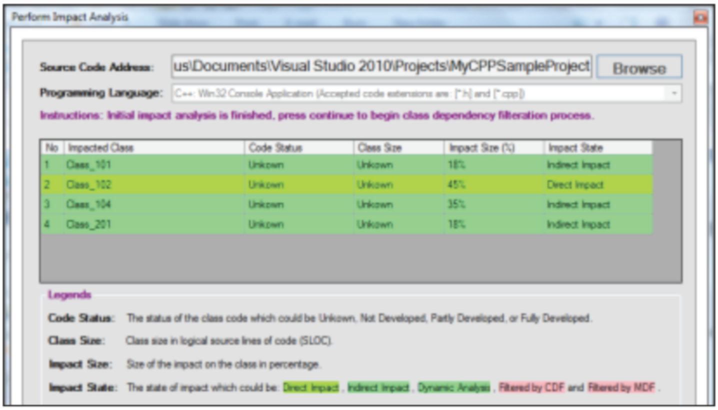

Figure 4. Sample of static impact analysis results

The next sub-process is filtering the static change impact analysis results using class dependency filtration (CDF). In CDF, each indirectly impacted class that has been identified by the static change impact analysis will be suspected. In this suspicion, a BFS search will be performed to dependent class artifacts tree. The goal of this search is to find a path from the indirectly identified impacted classes to the directly impacted classes; if such a path do not exists, this indirectly impacted class is considered as a false detection and it will be removed or filtered in the CDF results. Figure 5 explain the CDF concept.

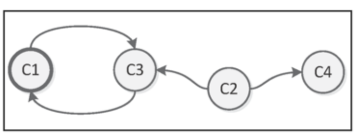

Figure 5. Sample of dependent directed graph

In Figure 5, C1 is a direct impacted class. C4 is considered as a false detection artifact because there is no path from C4 to C1. CDF will remove the C4. Figure 6 shows the sample of CDF results.

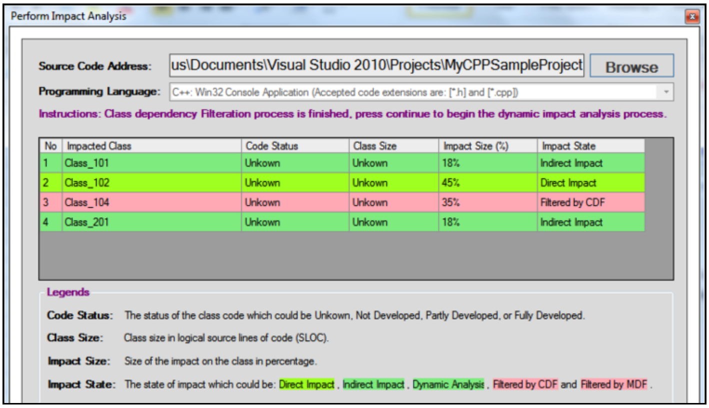

Figure 6. Sample of class dependency filtration results

The last sub-step is performing dynamic change impact analysis. This sub-step is a complex process. For simplicity, there are six sub-steps under this sub-step which are: (1) detect code status; (2) create actual method execution paths; (3) perform method dependency filtration (MDF); (4) perform method dependency addition (MDA); (5) perform dynamic change impact analysis; (6) update the change impact size.

The first sub-step, source codes will be inspected to find all the developed classes. Since our implementation scope is only C++ Win32 console applications, all the CPP (.cpp) and header (.h) files will be identified. Content of all files will be inspected to find if there exist any declarations of class from CIP model. Typical practice of C++ Win32 console application development, the programmer will declare all class names in header file and define them in the CPP file. However, it is also common that both declaration and definition of the class are written in the header file. Our code inspector will identify both types of class definition and declaration; categorize their development status; and save code information into the database. If the code file for a class is not found in the development directory, it will be considered as not developed class; but if it was found, another process will inspect the code to determine if the code is partially developed or fully developed.

Technically, we describe partially developed class as a class that contains stubs, fake methods, or incomplete methods in its current states of implementation. These partially developed methods could be detected by the code status tag; code status tag structure in C++ is as follow: [“///<status>” + Code Status + “</status>”]; where code status could be “Not Developed”, “Stubbed”, “Faked”, “Mocked”, “Partially Developed”, or “Fully Developed”. However, if the code status tag is not clearly defined, the code inspector will automatically detect as a partially developed method. At the same time, as the process is looking for the fully developed classes, it also detects fully or partially developed methods in all classes. The methods will be saved as method artifacts in the CIP artifacts database.

Furthermore, the tool also will calculate class size of fully developed classes in logical SLOC using pre-defined language specific rules. In C/C++, logical SLOC is calculated by counting the not-auto-generated codes containing preprocessor directives, terminal semicolons and terminal close-braces. A method is implemented in the code inspector of the tool, to first remove all comments from the codes and the calculate SLOC exactly based on the defined C/C++ language rules.

For each code inspection, a regular expression has been created to identify the match pattern for that inspection. Most modern programming languages support regular expressions to match a certain pattern in a text. The regular expressions used in the tool are shown in Table 1.

Table 1. Regular expression table

<table>
<tr><th>Regular Expression</th><th>Purpose</th></tr>
<tr><td>//\s*<status>\s*([\w\s]*)\s*</status></td><td>To get the code status.</td></tr>
<tr><td>::@MethodName\s*\((?:[\s\w]*)\)\s*
{\s*return\s*(null|\d|".*"|'.'|);</td><td>To detect faked methods which returns null, a number, a string, or a character in the first life of method.</td></tr>
<tr><td>::@MethodName\s*\((?:[\s\w]*)\)\s*
{\s*throw</td><td>To detect stubs that throws an exception in their first line.</td></tr>
<tr><td>(?:\.|->|::)([A-Za-z_][\w]*)\s*\(</td><td>To identify executed methods.</td></tr>
<tr><td>class\s+([A-Za-z_][\w]*)</td><td>To identify class declarations.</td></tr>
<tr><td>::([A-Za-z_][\w]*)\s*\((?:[\w\s]*)\)\s*{</td><td>To identify class definitions.</td></tr>
<tr><td>@ClassName\s*::\s*([A-Za-
_][\w]*)\s*\((?:[\w\s]*)\)\s*{</td><td>To identify methods definition of a declared class.</td></tr>
<tr><td>@ClassName\s*::\s*([A-Za-
z_][\w]*)\s*\((?:[\w\s]*)\)</td><td>To identify methods definition in class declaration.</td></tr>
<tr><td>\#include\s+(?:<|")([\w/.]+)(?:>|")</td><td>To find the include files</td></tr>
</table>

The next sub-step is to create method execution paths for all detected methods in fully developed classes. This sub-step reads the content of all methods and finds the methods executed by each identified method. The relation between methods and their executed methods will be saved to the CIP artifacts database to create the method execution paths. Based on the method executions paths that are created, two other sub-steps will be performed to improve the CIP model.

First, Method Dependency Filtration (MDF) will be performed to filter over-estimated interactions between the classes; a BFS search is used to find if there is any relation method between two interacting classes in CIP model. If there is no relation, the current relationship between two classes will be removed. Later, MDA will be performed to inspect whether if there exist actual interactions between two classes or not. If the MDA results show there relationship is exist, the new detected relationship will be added to the CIP model to become the new improved CIP model.

Lastly, the dynamic impact analysis sub-step will be performed on the improved CIP model. The dynamic change impact analysis will be performed by BFS search algorithm and new impact sizes will be determined as in static impact analysis. Figure 7 shows an example of dynamic analyses results in the tool.

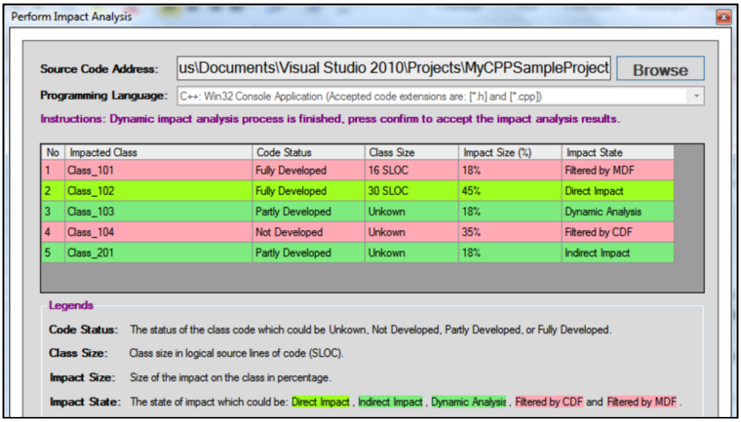

Figure 7. Sample of dynamic analysis results

2.4. Step 4: Estimating Change Effort

The final step of our model is to estimate the required change effort based on the change impact analysis results. To estimate the change effort, we use COCOMO II [23] effort estimation model as a basic reference. In our calculation, 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 in Eq. (1):

$$CPM = \left(\left(PM' - PM \right) + abs\left[\left(PM' - PM \right) \times DSF \right] \right) \times PR \tag{1}$$

Where,

- DSF is the development status factor based on Eq. (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 Eq. (2),
- 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. (2) to (4) show how PM′ is calculated. This equation will be justified with the assumption that the cost factors and the scale factors will not change with the change request. Accordingly, the mathematical justification for producing this equation is as follow:

$$PM' = \frac{PM'}{PM} \times PM \tag{2}$$

$$PM' = \frac{A \times CSize^{B} \times \left(\prod_{i=1}^{n} EM_i\right)}{A \times CSize^{B} \times \left(\prod_{i=1}^{n} EM_i\right)} \times PM \tag{3}$$

$$PM' = \left(\frac{CSize}{Size}\right)^{B} \times PM \tag{4}$$

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 Eq. (5),
- Size is the original estimation of code size,
- CSize is the estimated code size after implementing the change.

$$B = B_0 + B_1 \times \sum_{i=1}^{5} SF_i \tag{5}$$

Where,

- B_0 and B_1 are constant variables;
- SF_i 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 Eq. (4) 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 Eq. (6):

$$CSize = Size + \sum_{IC} \left(Size_{IC} \times ISF_{IC}\right) \tag{6}$$

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 Eq. (7).

$$ISF_{IC} = \sum_r P_{AT} \frac{CTF_r}{NR} \quad (7)$$

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,
- P_{AT}: A constant value for probability of change for affect type (AT) – where AT could be direct or indirect affection type),
- CTF_r: Change type factor based on the affected requirement change type which leads to the relation r. Due to space limitation, please refer our previous works in [11].

DSF in Eq. (1) 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. DSF will be calculated using the following Eq. (8):

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

Where,

- DSF stands for development status factor (DSF $\geq$ 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 which could be used to estimate multiplier values. Here is a sample of phase distribution weight of schedule and effort for a typical project using Rational Unified Process (RUP) [24]:

Table 2. Phase distribution weight in RUP

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

Table 2 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 as in Table 3:

Table 3. Estimated values for the multipliers

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

The last process of the model is to prepare its effort estimation results for the Change Control Board (CCB), which will be used to analyze the costs and impacts of change on the software. The results of the model are a set of prioritized impacted classes by their impact size, and the total effort required to implement the change. Figure 8 shows the implementation of change effort estimation in our tool and its sample results.

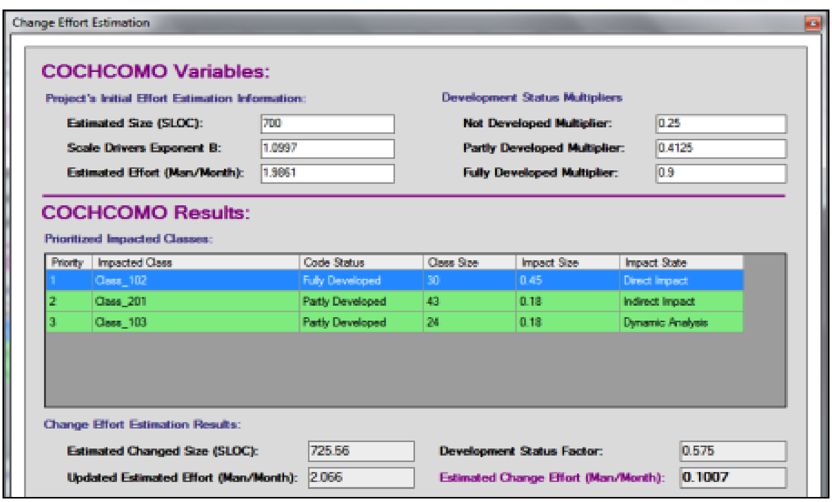

Figure 8. Sample of change effort estimation results

2.5. Step 5: Analyze Results

The final step of COCHCOMO tool is to automatically analyze the results of change impact analysis and change effort estimation. This analysis results will be used to evaluate the COCHCOMO model. The metrics used for analyzing change impact analysis results are Completeness, Correctness, and Kappa Value. Moreover, the metrics used for analyzing change effort estimation results are Relative Error (RE), Magnitude of Relative Error (MRE), and Mean Magnitude of Relative Error (MMRE). MMRE shows the overall accuracy of the COCHCOMO model for T number of change requests as tests. Figure 9 shows the implemented graphical user interface for analyzing the results with sample data. Also it is possible to print or save a report from this analysis results which could be used in evaluation.

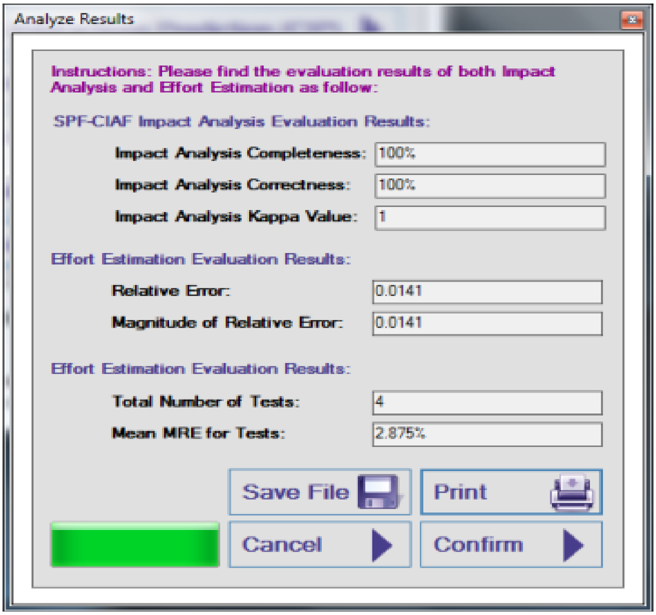

Figure 9. Sample of change effort estimation analysis results

3. Evaluation

This section describes the evaluation strategy that we have used to evaluate the accuracy of our change effort estimation tool. Due to space limitation, we only present our strategy on case study, change request data, evaluation procedure and evaluation metrics only.

3.1. Case Study

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

Table 4. Case studies

Case Study	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 partially 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.

3.2. Change Request

Considering four case studies (CS) with different development progress states, twenty change requests with different change types have been issued. Table 5 shows the distribution of the change requests.

Table 5. Change request distribution

Change Type	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

3.3. Evaluation procedure

The evaluation procedures are: (1) estimating change effort using tool; (2) performing actual change implementation to get the actual results; (3) measuring the accuracy of the estimated change effort results using evaluation metrics.

3.4. Evaluation metrics

Three effort estimation metrics were employed to measure the accuracy of the change estimation results. The metrics are Relative Error (RE) [25], Magnitude of Relative Error (MRE) [25], and Mean Magnitude of Relative Error (MMRE) [26].

RE: The RE shows the rate of relative errors and the direction of the estimation deviation [19]. A positive RE value matches to an under-estimate and a negative RE value to an over-estimate. The RE value is calculated as in Eq. (9):

$$RE = \frac{Actual\,\mathrm{Re}sults - Estimated\,\mathrm{Re}sults}{Actual\,\mathrm{Re}sults} \tag{9}$$

MRE: The MRE is similar to RE, but it is a metric for the absolute estimation accuracy only [25]. It calculates the rate of the relative errors in both cases of over-estimation or under-estimation as shown in Eq. (10):

$$MRE = abs\left[\frac{Actual\,\mathrm{Re}sults - Estimated\,\mathrm{Re}sults}{Actual\,\mathrm{Re}sults}\right] \tag{10}$$

MMRE: The MMRE or the Mean Magnitude of Relative Error is the percentage of average of the MREs over an entire data set [26]. It is used for calculating the accuracy of an estimation technique using t number of tests as it is shown in Eq. (11):

$$MMRE = \frac{100}{t}\sum_{i}^{t} MRE_i \tag{11}$$

The RE and MRE metrics will be calculated for each predicted impacted class from the change request experience to measure the accuracy of the change impact size estimation in our approach. But the MMRE will be calculated for the whole case study, which contains ten change requests and several impacted classes.

The results of our approach are more accurate when the MMRE values are smaller. Study by [27] stated that in estimation models, if MMRE value is less than 25% then the estimation results is considered as an accurate estimation model. Therefore, if the MMRE values calculated from results of estimation in the approach are less than 25%, the proposed approach will be proved to be acceptably accurate.

4. Results and Discussion

In this section the change effort estimation and error analysis results for all case studies (CS1 to CS4) are demonstrated.

Table 6. Evaluation results of RE and MRE for all case studies

Case Study	CR No	Change Effort		Tool Evaluation	
		Actual	**Estimated**	**RE**	**MRE**
CS1	CR1	0.17619	0.1439	0.18327	0.18327
	CR2	0.1924	0.1696	0.11850	0.11850
	CR3	0.0078	0.0055	0.29487	0.29487
	CR4	-0.0478	-0.0401	0.16109	0.16109
	CR5	-0.9108	-0.9034	0.00812	0.00812
	MMRE				**15.32%**
CS2	CR6	0.1600	0.1626	-0.01625	0.01625
	CR7	0.24107	0.2003	0.16912	0.16912
	CR8	0.00378	0.0059	-0.56085	0.56085
	CR9	-0.03986	-0.0481	-0.20672	0.20672
	CR10	-0.69118	-0.6876	0.00518	0.00518
	MMRE				**19.16%**
CS3	CR11	0.2248	0.2187	0.02714	0.02714
	CR12	0.1924	0.1851	0.03794	0.03794
	CR13	0.0253	0.0109	0.56917	0.56917
	CR14	-0.0319	-0.0251	0.21317	0.21317
	CR15	-0.3901	-0.394	-0.01000	0.01000
	MMRE				**17.15%**
CS4	CR16	0.2900	0.3281	-0.07117	0.07117
	CR17	0.3227	0.2578	0.20112	0.20112
	CR18	0.0125	0.0084	0.32800	0.32800
	CR19	-0.0065	-0.0064	0.01538	0.01538
	CR20	-0.1109	-0.1205	-0.08656	0.08656
	MMRE				**14.04%**

To recap, the evaluation has been focusing on comparing results between the estimated change effort with the actual change effort. We have used the MMRE as the comparison metric.

According to [28] 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 [29], 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%.

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 7 shows the MRE and MMRE of change requests in each case study.

Table 7. MRE and MMRE based on change types (CT) across case study (CS)

CT	CS1	CS2	CS3	CS4	Average
CT1	0.1832	0.0162	0.0271	0.0711	0.074458
CT2	0.1185	0.1691	0.0379	0.2011	0.131670
CT3	0.2948	0.5608	0.5691	0.3280	0.438223
CT4	0.1610	0.2067	0.2131	0.0153	0.149090
CT5	0.0081	0.0051	0.0100	0.0865	0.027465
MMRE	15.3%	19.2%	17.2%	14.0%	16%

Due to space limitation, a quick look on the average MMRE value revealed that: (1) our proposed tool has 16% relative error on average which is better than our expectation; and (2) all MMRE values for the case studies is less than 20%. This analysis indicated that the proposed tool is acceptably accurate.

5. Conclusion

We have developed a change effort estimation tool that utilizes our previous work on change impact analysis. This tool is meant to support the implementation of our previous change effort estimation model. Basically, the tool automates both change impact analysis and change effort estimation processes that exist in the model.

The tool was evaluated by four different case studies over twenty change requests. The presented evaluation results demonstrate that the tool produces a reliable change effort estimation results. Our proposed tool has 16% relative error on average which is better than our expectation and all MMRE values for the case studies is less than 20%. This analysis indicated that the proposed tool is acceptably accurate.

The results of this paper are part of an ongoing research to over-come 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 tool by considering more change requests from different case studies. Also, we will extend this tool for change cost estimation as well.

References

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

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

[3] A. Finkelstein, J. Kramer, Software Engineering: A Roadmap, *Proceedings of the Conference on the Future of Software Engineering*, pp. 3-22, 2000

[4] F.P. Brooks, No silver bullet. *IEEE Computer*, no. 1, Jan.-Feb. 2008., vol. 25, pp. 91-94, 1987.

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

[6] 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, 1993.

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

[8] A. Bianchi, A.R. Fasolino, G. Visaggio, An Exploratory Case Study of Maintenance Effectiveness of Traceability Models, *Proceedings of the 8th International Workshop on Program Comprehension*, pp. 149-158, 2000.

[9] N. Kama, Integrated change impact analysis approach for the software development phase, *International Journal of Software Engineering and its Applications*, Volume: 7 Issue: 2 Pages: 293-304 Published: 2013

[10] N. Kama, F. Azli, A change impact analysis approach for the software development phase, *Proceedings - Asia-Pacific Software Engineering Conference, APSEC*, Volume: 1 Pages: 583-592 Published: 2012

[11] M.H. Asl, N. Kama, A change impact size estimation approach during the software development, *Proceedings of the Australian Software Engineering Conference, ASWEC* Pages: 68-77 Published: 2013

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

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

[14] Z. Yinhuan, W. Beizhan, Z. Yilong, 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, pp. 941-943, 2009.

[15] C.A.L. Garcia, 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, pp. 820-825, 2008.

[16] V. Nguyen, L. Huang, B. Boehm, An analysis of trends in productivity and cost drivers over years. *Proceedings of the 7th International Conference on Predictive Models in Software Engineering*, Banff, Alberta, Canada: pp. 1-10, 2011.

[17] I. Attarzadeh, A. Mehranzadeh, 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 167-172, 2012.

[18] G.R. Finnie, G.E. Wittig, 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, pp. 281-289, 1997.

[19] S. Grimstad, 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: pp. 58-65, 2006.

[20] N. Kama, S. Basri, Extending Change Impact Analysis Approach to Support Change Impact Size Estimation for Software Development Phase, *2014 6th International Conference on Computer Research and Development (ICCRD 2014),* Hanoi, Vietnam, in-press.

[21] N. Kama, F.A.A. Ridzab, Requirement Level Impact Analysis with Impact Prediction Filter, *4th International Conference on Software Technology and Engineering (Icste 2012)*, Pages: 459-464 Published: 2012, ASME
[22] R. Zhou, E.A. Hansen, Breadth-first heuristic search. *Artificial Intelligence*, vol. 170, no. 45, pp. 385–408, 2006.
[23] B. Sharif, S.A. Khan, 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, pp. 170-174, 2012.
[24] P. Kruchten, The Rational Unified Process: An Introduction, Addison-Wesley, 2004.
[25] M. Jøgensen, 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, pp. 993–1007, 2004.
[26] V. Nguyen, B. Steece, B. Boehm, A constrained regression technique for cocomo calibration, *In Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement (ESEM '08)*, ACM, New York, NY, USA, 213-222., 2008.
[27] L. Huang, Y.-T. Song, Precise dynamic impact analysis with dependency analysis for object-oriented programs, *5th ACIS International Conference on Software Engineering Research, Management Applications, 2007*, SERA 2007., aug. 2007, pp. 374 –384, 2007.
[28] S. Basha, D. Ponnurangam, Analysis of Empirical Software Effort Estimation Models, *International Journal of Computer Science and Information Security (IJCSIS)*, Vol. 7, No. 3, 2010.
[29] S.-J. Huang, N.-H. Chiu, L.-W. Chen, Integration of the grey relational analysis with genetic algorithm for software effort estimation, *European Journal of Operational Research*, 188(3), 898-909, 2008.