A Change Impact Size Estimation Approach during the Software Development

Mehran Halimi Asl, Nazri Kama
Advanced Informatics School
Universiti Teknologi Malaysia
Kuala Lumpur, Malaysia
Email: mehran.net1@gmail.com, nazrikama@ic.utm.my

***Abstract*—Modern software development is iterative and encourages frequent interactions between the software development team and the stakeholders of the software. These interactions will generate change requests as the requirements gradually evolve to meet the stakeholders expectations or due sudden shifts in circumstances. However, during the development, software developers have to assess the impact of these change requests with respect to incomplete status of software artefacts. This situation presents one important aspect when considering a change request, which is how to estimate the size of change impact that is required by a requirement change, given inconsistent states of artefacts across the project. Therefore, this paper introduces a new Change Impact Size Estimation (CISE) approach for the software development phase. Further on that, a prototype tool has been developed to support the implementation of the approach and evaluation. The case study method used for evaluating this approach corroborated the functionality and accuracy of this approach for estimating the change impact size.**

***Index Terms*—impact analysis; change impact size; class interactions prediction; requirement change; software change management; software development phase**

I. Introduction

Software undergoes changes not only in the maintenance phase, but also throughout all the software development phases, including design and implementation. In other words, changes could happen at all stages of software development life cycle (SDLC). On the one hand, accepting too many changes causes additional cost and delay in the completion. On the other hand, rejection of the changes may cause customer dissatisfaction. Consequently, it is essential for the software project managers to make effective decisions on change acceptance specially during software development to keep the evolving needs of the customer satisfied.

The change management processes used in industry are commonly a tailored version of one of the famous change management processes. The Sommerville [1] change management process is one of the most well-known processes in change management field. He explained, after capturing the change request by a change request form, how the change should be validated and accepted. One of the change request attributes used in the change request form is affected components or affected classes. This process highlights the role of change impact analysis and its critical task for having a good prediction of directly and indirectly affected software artefacts before accepting the change.

Normally the affected artefacts are predicted by a method called change impact analysis. However, the number of affected artefacts is not the only relevant reason for change acceptance, but also the complexity of the change have to be determined. An important attribute of change complexity used for the change acceptance decisions is the estimation of change impact size through the prediction of impacted artefacts. This estimation is performed differently in the software development phase compared to the software maintenance phase. For the software development phase, the prediction that is made by the software project manager have to assess the impact of the change request with respect to incomplete status of software artefacts in contrast with the software maintenance phase. This situation indirectly presents one important aspect when estimating change impact size for the software development phase which is how we estimate the change impact size, given inconsistent states of artefacts across the development of software. In other words, estimating the magnitude of change impact on the software artefacts which are under development has became a challenge [2]. Predicting the change size of impacted classes is not only important for change acceptance decisions, but also for preventing more dangerous faults after implementing the changes [3]. The change impact size shows the proportion of change in impacted artefacts, which will increase the usefulness and effectiveness of impact analysis results in change management.

Accordingly, many researches tried to overcome this challenge. Sneed [4] performed a research on how to estimate the size and complexity of requested changes in web applications. His approach was to measure the size of web application, then detecting the domains which will be impacted by the change. Despite the promising results produced by his method, our goal is to develop a more general size estimation approach; not only for maintaining a certain type of web applications. Additionally, Murat Kahraman [5] developed a method to estimate the impact size more generally. However, both of these methods are developed for fully developed codes without considering the not developed or partly developed codes in the software development phase. In this empirical study, we develop and implement a new general approach for estimating change impact size which applicable during software development phase.

This paper introduces a new Change Impact Size Estimation

978-0-7695-4995-8/13 $26.00 © 2013 IEEE
DOI 10.1109/ASWEC.2013.18

approach or also known as CISE. This approach utilizes the integrated change impact analysis technique proposed by Nazri Kama [6], [7], [8]. The term "integrated" is used here because this technique integrates between the static analysis and the dynamic analysis techniques to perform change impact analysis for the software development phase. This integrated change impact analysis technique has been tested and evaluated against similar methods in our previous research [9], and it proved to be accurate and effective during the software development phase.

Moreover, in this research, we extended the capabilities of this technique by incorporating a new calculation method to support change impact size estimation for a particular change request. In other words, the new change impact size estimation approach is a combination between the integrated change impact analysis technique and a new change impact size estimation method.

This paper is laid out as follow. First, we explained the integrated impact analysis approach in Section II. Then, we describe our classification of the requirement changes which affect the change impact size in Section III. Our approach for estimating the change impacts and their change size is introduced in Section IV. The evaluation procedure of the approach is explained in Section V. Finally, we describe our conclusions and future works in Section VI.

II. Change Impact Analysis

Change impact analysis, or just impact analysis, is the method of predicting the impacts of the requirement change on the software elements [10]. The impacted software elements could be design artefacts such as packages and design classes, coding artefacts like components and classes, testing artefacts such as test cases and test reports, or any other software elements and documents. Impact analysis clarifies the consequences or the effects of the requested changes on the software codes and documentations. In addition, it can help the software project managers through the decisions to accept, modify, defer, or even reject the change requests based on their predicted consequences.

Change impact analysis is performed by tracing the affected requirements by change requests to the impacted software artefacts and codes before they are implemented. This oversight of change impacts can help the software development and maintenance to control and manage the changes greatly. There are two approaches of impact analysis, which are dependency analysis and traceability analysis [6].

Dependency analysis also known as program analysis is the analysis of relations only between source codes by exploring the organization of the codes [11]. Traceability analysis uses the relationships between software artefacts for analysis; including requirements, design artefacts, source codes, and test artefacts; across different software phases [12].

Based on the impact analysis approaches, there are also two categories of impact analysis techniques. The impact analysis techniques are static analysis techniques which are established by traceability analysis [13]; and dynamic analysis techniques which are established by dependency analysis [14]. Also another type of impact analysis techniques exist which is the integrated impact analysis. It is a combination of other two categories [8], [15], [16].

A. Static Impact Analysis

The static impact analysis technique analyses static information from software artefacts such as requirements, designs, classes, and tests to create a set of potential impacted classes [13], [6]. Two categories of static model information models can be used to perform static impact analysis techniques, which are high level artefacts and low level artefact. The low level artefact model uses source code model by reverse engineering from the existing source code or class artefacts to identify the set of potential impacted classes. On the other hand, high level artefact model uses the design and requirement artefacts to identify the set of potential impacted classes.

B. Dynamic Impact Analysis

The dynamic analysis technique analyses dynamic information created by executing the code to create a set of potential impacted classes [14]. In dynamic analysis, the dynamic information is generated by parsing through the source code, and its outcome is highly dependent on the scenarios selected for executing the code. There are two categories of dynamic information to perform dynamic analysis which are the class dependency graph [16]; and the method execution paths [11].

C. Integrated Impact Analysis

For the software development phase, the prediction that is made has to assess the impact of the change request with respect to incomplete status of software artefacts compared to the software maintenance phase. Further studies [8], [15], [16], show that combining static analysis and dynamic analysis techniques into one integrated technique can produce more reliable impact analysis results. Earlier researches [8], [9] illustrated that integrated impact analysis would be a better approach for impact analysis software development phase.

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 artefacts for impact analysis. This framework [8] is called Software Development Phase Change Impact Analysis Framework (SDP-CIAF). The SDP-CIAF [8] 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 [6] using requirement and design artefacts; and in stage two the potential impacted classes will be identified from the relations in the CIP model. At last the over-estimated results will be filtered and using filtration techniques to determine the final impact analysis results.

III. Types of Requirement Change

Software requirements are the clarifications of the stakeholders needs for software engineers. They are identified by a well-known process in software engineering which is requirement engineering. However, requirements are usually subject to change in both during development and operation phases. There are two main categories for software requirements according to Leffingwell and Widrig [17]: (1) functional and (2) non-functional. Functional requirements capture and describe system functionality, whereas non-functional requirements capture all other needs from software product such as quality, portability, performance, reliability and usability. In this research, the impacts of change in functional requirements which have impact on the software functionalities are examined.

Requirement change, also known as requirement volatility, is the source of change in any software change specification. It is the trend of the evolving needs of stake holders and the evolution of software environment [18]. Nurmuliani [19] has investigated the impact of three requirement change types which are deletion, modification, and addition. Additionally, we extended this classification further and expanded the modification possibilities.

Shao *et al.* [3] study, indicates that changes size has direct relation to changes impacts, and moreover small changes has lesser impact than larger changes. Therefore, change types in our study are based on the size of the requested change. We classified the requirement change types that affect the change impact size in nine change types, each change type has a change type factor (CTF) value according to the nature of the requirement change type. The 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 change impact size. Table I shows the requirement change types and their nominated CTF values. Furthermore, here are a set of brief descriptions for the effect of each requirement change type:

1) Addition: It is a new requirement, and all new code will be added to the original code.
2) Modification-Major Grow: Three quarter of the estimated impact code will be added to the original code.
3) Modification-Grow: Only half of the estimated impact code will be added to the original code.
4) Modification-Minor Grow: One quarter of the estimated impact code will be added to the original code.
5) Modification-Negligible: The change is so insignificant that code size will not be changed much.
6) Modification-Minor Shrink: One quarter of the estimated impact code will be removed from the original code.
7) Modification-Shrink: Only half of the estimated impact code will be removed from the original code.
8) Modification-Major Shrink: Three quarter of the estimated impact code will be removed from the original code.
9) Deletion: The requirement will be deleted, and the codes generated by this requirement will be removed from the original code.

TABLE I
Requirement Change Types

Change Type	CTF
Addition	1.00
Modification-Major Grow	0.75
Modification-Grow	0.50
Modification-Minor Grow	0.25
Modification-Negligible	0.10
Modification-Minor Shrink	-0.25
Modification-Shrink	-0.50
Modification-Major Shrink	-0.75
Deletion	-1.00

In our specification of change request, the changed requirements are the most critical change request attributes. They are known as the directly affected requirements that will be used to find the impacted classes by performing impact analysis on the CIP model. Each affected requirement should have a change type which could be one of the change types in Table I. These change types elucidate the effect of requirement change on the code. This specification of change types and their CTF values are defined based on typical change requests on our projects. In addition, the author emphasize that the described classification of change types is only one way of quantizing the change types, other classifications like two-third are also possible. Hence, they can be redefined for change requests of any software organization. It is crucial to first understand the change request, and second to determine the implication of the change requests on requirements before performing the change impact analysis. It should be noted that CTF value only shows the proportion of the change on a number of requirements, it doesn't indicate the change size. For example, adding an insignificant requirement to the software will only change a very small piece of code, but it will change it greatly.

IV. A Change Impact Size Estimation Approach

This section describes our proposed approach for estimation the change impact size which we name as CISE. In this approach, the goal is to only predict the impacted classes and then estimate their change size. Other related impacted artefacts can be determined according to their traceability relations to the classes. The CISE approach performs change impact analysis on the directly affected requirements. Afterwards it estimates the change impact size for each predicted impacted classes by their requirement change types, which were described in Section III.

The process of CISE approach for change impact size estimation includes three steps as in Fig. 1. This process starts by developing the CIP model from requirement and software design artefacts. Afterwards, it performs change impact analysis using the integrated impact analysis approach to predict the impacted classes. Finally, the change impact size for each impact classes will be estimated. The final results of this process are a set of prioritized impacted classes and their estimated change size in percentage. These predicted results

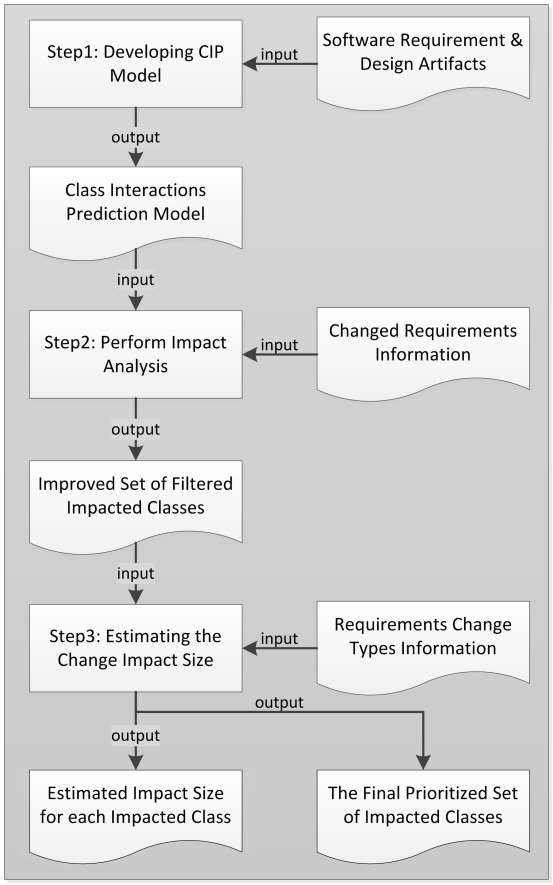

Fig. 1. CISE Approach Overview

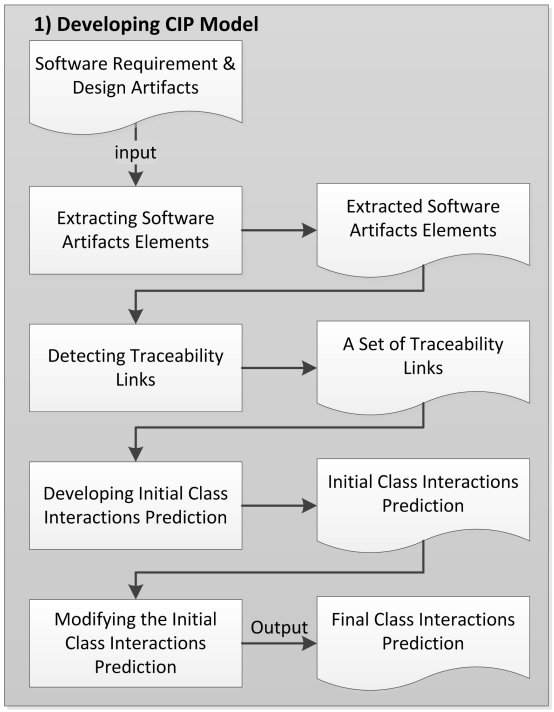

Fig. 2. Developing CIP Model Process Flow

together are used to estimate the size of the change impact on the software artefacts.

A. Step 1: Developing CIP Model

The first step in CISE approach is to develop the Class interactions Prediction (CIP) model. The CIP model contains the vertical and horizontal relationships between requirements and design artefacts. Vertical relationships are referred to the relations between same kind of artifices; and horizontal relationships are referred to the relations between one types of artefact to the other types. The development of the CIP model can be automated by a predictive technique, which has to be able to predict the artefact interactions. This technique predicts the CIP interactions using two analysis approaches which are: (1) significant object interactions analysis in the requirement artefact; (2) design patterns analysis in the design artefact. This technique is explained in our previous works [6], [20], [7], and here we only mention its processes briefly.

As it is shown in Fig. 2, the development of CIP model has four processes which are: “Extracting Software Artefact 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 their results will provide the inputs for the next one. In the end, the final class interactions prediction model will be developed by the last process. This CIP model contains the requirement and design artefacts information as well as the information about the relations between them. It will be used to perform impact analysis in the next stage.

B. Step 2: Preforming Impact Analysis

Change Impact analysis will be performed on the developed CIP model from the previous step. In this step integrated change impact analysis will be performed, and afterwards its results will be filtered 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 which are Class Dependency Filtration (CDF) and Method Dependency Filtration (MDF). The process of the change impact analysis used here is described in [6], and it has recently been evaluated against similar techniques in [9]. The evaluation results show that this impact analysis technique is more superior for the software development phase in compare to other similar methods.

Moreover, the impact analysis method used in CISE approach is adopted from our previous works. Our impact analysis process begins by performing static impact analysis on the developed CIP model to find the direct and indirect affected classes. The static impact analysis will firstly find the directly impacted classes, which are the first layer of classes affected by a particular changed requirement without

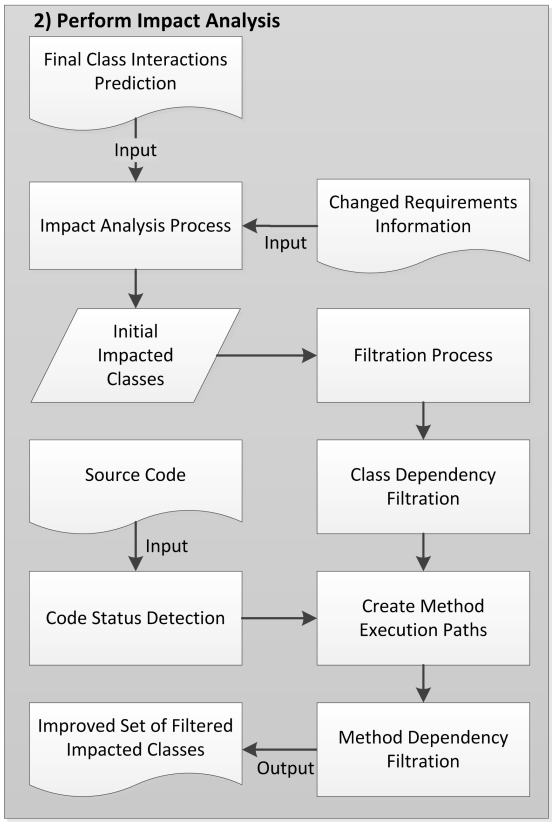

Fig. 3. Preforming Impact Analysis Process Flow

vertical traceability relations consideration. Then indirectly impacted classes will be identified by complete traceability search through the CIP interactions to find all the related classes to the changed requirement. Moreover, Fig. 3 shows the complete process flow of the impact analysis step in CISE method.

The implementation of static impact analysis in CISE uses a Breadth-first search (BFS) algorithm [21] to search for the impacted classes in the CIP model. Where, the software artefacts in CIP model are 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 CDF. The CDF process finds the potential but falsely identified impacted classes by tracing the flow of interactions between the classes.

In CDF process, 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 for the dynamic impact analysis procedure.

In the dynamic impact analysis procedure, we aim to find the actual interactions between the fully developed classes, and improve CIP model by adding the method interactions to the CIP model. Then the overestimated static impact analysis results are removed by MDF filtration.

The most demanding challenge in this process is how to detect fully developed classes. To overcome this challenge we managed to develop a mechanism to identify the status of the codes. In our approach, code status could be not developed, partly developed or fully developed. It must be also noted that the level of completeness is not relevant, our aim is to only identify the fully developed classes. 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, using a special tag for classes and methods is recommended to keep its code status as we develop the program.

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.

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

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

3) Incomplete Method: If a method is not stub or fake, and still, it does not perform its functionality fully, it is considered as incomplete. The methods hierarchies from the designs are compared with their actual call hierarchy in codes. If they are not the same, this method is not completely developed yet. This approach for detecting incomplete methods could only

be implemented if all the information about the class methods and their relations are available in the early design documents and in the CIP model.

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

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

C. Step 3: Estimating the Change Impact Size

At last, the change impact size for all predicted impacted classes are calculated in Step 3. Change impact size is calculated by impact size factor (ISF), which is the proportion of change in percentage that will be used to predict the size of code after the change is implemented.

ISF is the sum of probable change volumes in each identified impacted class according to the change type factor for the related requirement change requests. It is possible to have a prediction of the change impact size for each identified impacted class by ISF. The equation used in CISE approach to calculate ISF can be seen in (1).

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

Where ISF_{IC} is the impact size factor for impacted class (IC); r stands for relation from requirement to the impacted class; NR is the number of requirement artefacts that have relation to the impacted class; PV_{AT} is a constant value for probability of change volume of the affect type (AT); and CTF_r is the change type factor based on the affected requirement change type (see Table I) which lead to the relation r.

Based on the traceability technique used for the change impact analysis, the affect types (AT) can be direct or indirect. Only the impacted classes which have been identified by the impact analysis technique directly from the changed requirement without tracing its interactions with other requirements are considered as directly affected classes. This value is constant for the project and determined by the development history of the team or try and error method. In addition, usually the P_{AT} for indirectly impacted classes should be less than direct impacted classes. These values are highly dependent to the development model used, and should be calibrated for the project.

The implementation of ISF calculation finds all the relation paths to the impacted classes, and identifies the changed type of the changed requirements which the impacted classes are originated from. Then it calculates the ISF value for each

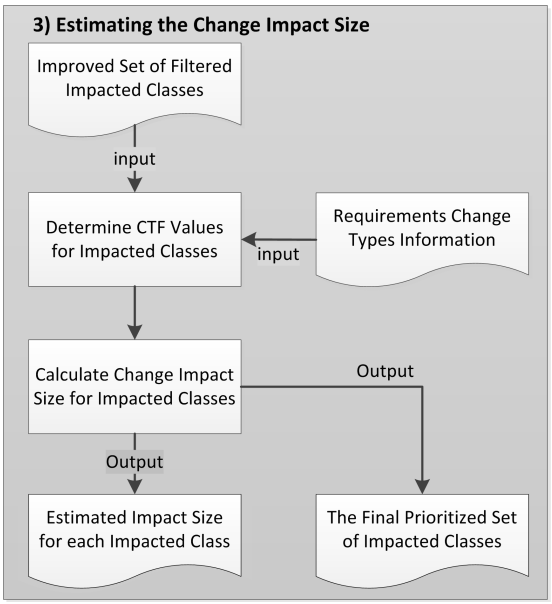

Fig. 4. Estimating the Change Impact Size Process Flow

impacted class as we discussed before. Although, ISF is just a rough estimation, and it is not very accurate, it can produce an overall acceptable prediction of the future class size expansion after the change with calibrated variables. Fig. 4 shows an overview of change impact size estimation process flow.

The final results of the CISE approach are the set of prioritized impacted classes by their impact and their impact sizes. Moreover, the code size after change can also be estimated by multiplying the ISF value for the impacted class to its size. The size of impacted classes could be measured if all the impacted classes are fully developed, or the class size estimations are available. Measuring or estimating the size of codes are out of the scopes of this research, but for the implementation purposes, a method for measuring the size of fully developed classes in source lines of code (SLOC) in C++ [22] is implemented. This method implies that certain items should be counted in the code to measure the logical source lines of code, and it assures that the logical SLOC of the code would be almost the same with different styles of coding. In the next section, the methods used for evaluating the CISE approach and measuring its accuracy as well as the analysis of the evaluation results will be described.

V. Evaluation

For evaluating this empirical research results and measuring the accuracy of the proposed approach an explanatory case study evaluation method [23] with several controlled change experiences is used. According to the guidelines described in Runeson and Höst [23] study, the five steps for evaluation with case study method are: (1) Design, (2) Preparation, (3) Collection, (4) Analysis, and (5) Report. These steps have been performed thoroughly according to the guidelines.

TABLE II
CHANGE REQUESTS TYPES

CR No	Change Type
CR1	Addition
CR2	Modification-Grow
CR3	Modification-Negligible
CR4	Modification-Shrink
CR5	Deletion
CR6	Modification-Grow & Shrink
CR7	Modification-Grow & Negligible
CR8	Addition & Modification-Negligible
CR9	Deletion & Modification-Shrink
CR10	Deletion & Addition

A. Case Study Design

For the evaluation an easy to understand software development project which has been developed by Rational Unified Process (RUP) methodology is chosen as the case study. The selected project is On-Board Automobile (OBA) project. The purpose of this development project is to develop a simulation for an auto cruise system on a vehicle. This project has been designed and developed by master of software engineering students in Advanced Informatics School (AIS) faculty at UTM University. For implementation of this project Visual Studio 2010 and Win32 C++ console application have been used. This console application is the logic core of vehicle simulation software to control the behaviours of a simulated vehicle while it is auto cruising.

There have been totally four students and a project manager from the faculty staffs involved with development of OBA project. Its designs contain fifty three requirements within five use cases and fourteen design classes within eight packages. We had access to all the requirements and designs of this project, which were documented during its development using DOD standards.

Considering the requirement change types in Table I, ten change requests have been picked. These change request will experience all the five basic requirement change types, as well as five composition of multiple requirement change types as they are shown in Table II. For example the change request CR10 contains both addition and deletion requirement change types, in which a set of requirements will be replaced by another requirements. The results of the change experiences will be used for analysis and evaluation of CISE approach.

B. Evaluation Metrics

For evaluating the accuracy of CISE approach, three metrics for measuring the accuracy of estimations have been selected. These metrics are usually used for evaluating effort estimation techniques, but they are also suitable for the purposes of this study. The three selected metrics are Relative Error (RE) [24], Magnitude of Relative Error (MRE) [24], and Mean Magnitude of Relative Error (MMRE) [25].

1) RE: The RE shows the rate of relative errors and the direction of the estimation deviation [24]. 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 (2).

$$RE = \frac{(ActualResult - EstimatedResult)}{(ActualResult)} \quad (2)$$

2) MRE: The MRE is similar to RE, but it is a metric for the absolute estimation accuracy only [24]. It calculates the rate of the relative errors in both cases of over-estimation or under-estimation as shown in (3).

$$MRE = abs[\frac{(ActualResult - EstimatedResult)}{(ActualResult)}] \quad (3)$$

3) MMRE: The MMRE or the Mean Magnitude of Relative Error is the percentage of average of the MREs over an entire data set [25]. It is used for calculating the accuracy of an estimation technique using T number of tests as it is shown in (4).

$$MMRE = \frac{100}{T}\sum_{i}^{T} MRE_i \quad (4)$$

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 CISE 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 as possible. Huang *et al.* [14] 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 CISE approach were less than 25%, the proposed CISE approach will be proved to be acceptably accurate.

C. Results and Analysis

The CISE approach was implemented in an automated prototype tool according the descriptions in Section IV. This tool has been used to perform integrated impact analysis on the developed CIP model and to estimate the change impact size of the predicted impacted classes from the requested changes.

Moreover, the requested changes had been implemented on the software by a member of the original development team who has worked on these projects to identify the actual impacted classes and their change size. In fact, the original OBA development team has helped us greatly through this research. They have helped us to collecting the actual results, validating the CIP model, and the results of this research. Afterwards, these actual results have been compared with the estimated results to measure the relative errors of CISE approach results.

The purpose of this evaluation is to measure the accuracy of the change impact size estimation, and the results of the change impact size estimation are dependent on the results of change impact analysis in our approach. Therefore, the CIP model was validated to be completely true to ensure that the results of the integrated change impact analysis are correct and complete.

All the results of change impact analysis and their estimated change impact size are gathered and analysed. However, demonstrating all these results and analysis in this paper is not feasible. A brief description of these results is demonstrated in this section.

Table III demonstrates the analysis of the change impact size estimation results on the OBA case study. This table shows the impacted classes by all the change requests and their estimated change impact factor. It also includes the calculated RE and MRE values for all correctly identified impacted classes. The calculated RE values from Table III have been analysed to evaluate the accuracy of CISE approach. A scatter chart from the RE values (see Fig. 5) and a line chart from MRE values (see Fig. 6) have been developed to demonstrate the data analysis.

Fig. 5 illustrates that none of the RE values were more than 0.25 or less than -0.25. This RE value indicates that the relative error was less than 25%. Thus, it has not exceeded its evaluation criterion. Further investigation in these data revealed a small weakness of our approach in measuring the change size for indirectly impacted classes which has affected

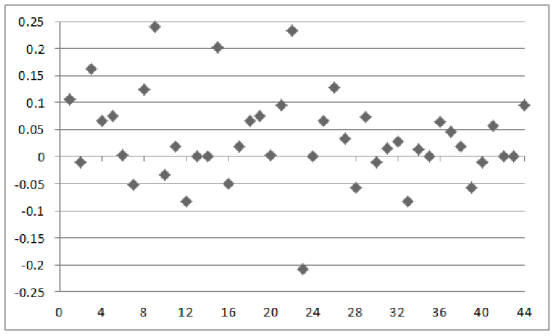

Fig. 5. Scatter Chart for RE Values

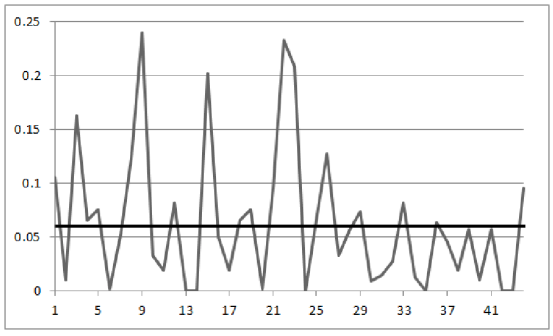

Fig. 6. Line Chart for MRE Values

TABLE III
OBA CASE STUDY RESULTS ANALYSIS

CR	Impacted Class Name	ISF%	RE	MRE
CR1	Vehicle	4.65%	0.1058	0.1058
CR1	AutoCruiseCtrl	6.06%	-0.0100	0.0100
CR1	DriverBny	3.77%	0.1622	0.1622
CR2	AutomobileEngine	6.25%	0.0658	0.0658
CR2	Vehicle	3.57%	0.0751	0.0751
CR2	AutoCruiseCtrl	4.69%	0.0021	0.0021
CR2	OBATimer	3.26%	-0.0516	0.0516
CR2	DriverBny	2.88%	0.1237	0.1237
CR3	ControlPanel	0.19%	0.2400	0.2400
CR3	AutoCruiseCtrl	0.31%	-0.0333	0.0333
CR4	RemindMaintenanceCtrl	−5%	0.0196	0.0196
CR5	Vehicle	−16.67%	-0.0818	0.0818
CR5	Trip	−35%	0.0000	0.0000
CR5	CalculateFuelConsumptionCtrl	−100%	0.0000	0.0000
CR5	ControlPanel	−13.21%	0.2015	0.2015
CR5	DriverBny	−13.46%	-0.0497	0.0497
CR6	RemindMaintenanceCtrl	−5%	0.0196	0.0196
CR6	AutomobileEngine	6.25%	0.0658	0.0658
CR6	Vehicle	3.57%	0.0751	0.0751
CR6	AutoCruiseCtrl	4.69%	0.0021	0.0021
CR6	OBATimer	2.17%	0.0958	0.0958
CR6	DriverBny	1.92%	0.2320	0.2320
CR7	ControlPanel	3.02%	-0.2080	-0.2080
CR7	AutoCruiseCtrl	5%	0.0000	0.0000
CR7	AutomobileEngine	6.25%	0.0658	0.0658
CR7	Vehicle	3.81%	0.1275	0.1275
CR7	OBATimer	3.48%	0.0333	0.0333
CR7	DriverBny	3.08%	-0.0566	-0.0566
CR8	Vehicle	4.88%	0.0740	0.0740
CR8	AutoCruiseCtrl	6.36%	-0.0095	-0.0095
CR8	DriverBny	3.96%	0.0152	0.0152
CR8	ControlPanel	3.89%	0.0275	0.0275
CR9	Vehicle	−16.67%	-0.0818	0.0818
CR9	Trip	−37.5%	0.0131	0.0131
CR9	CalculateFuelConsumptionCtrl	−100%	0.0000	0.0000
CR9	ControlPanel	−14.15%	0.0641	0.0641
CR9	DriverBny	−14.42%	0.0457	0.0457
CR9	RemindMaintenanceCtrl	−5%	0.0196	0.0196
CR10	Vehicle	−11.63%	-0.0573	0.0573
CR10	AutoCruiseCtrl	6.06%	-0.0100	0.0100
CR10	DriverBny	−9.43%	0.0570	0.0570
CR10	Trip	−35%	0.0000	0.0000
CR10	CalculateFuelConsumptionCtrl	−100%	0.0000	0.0000
CR10	ControlPanel	−9.26%	0.0952	0.0952
Number of Impacted Classes: 44			**MMRE:**	6.30%

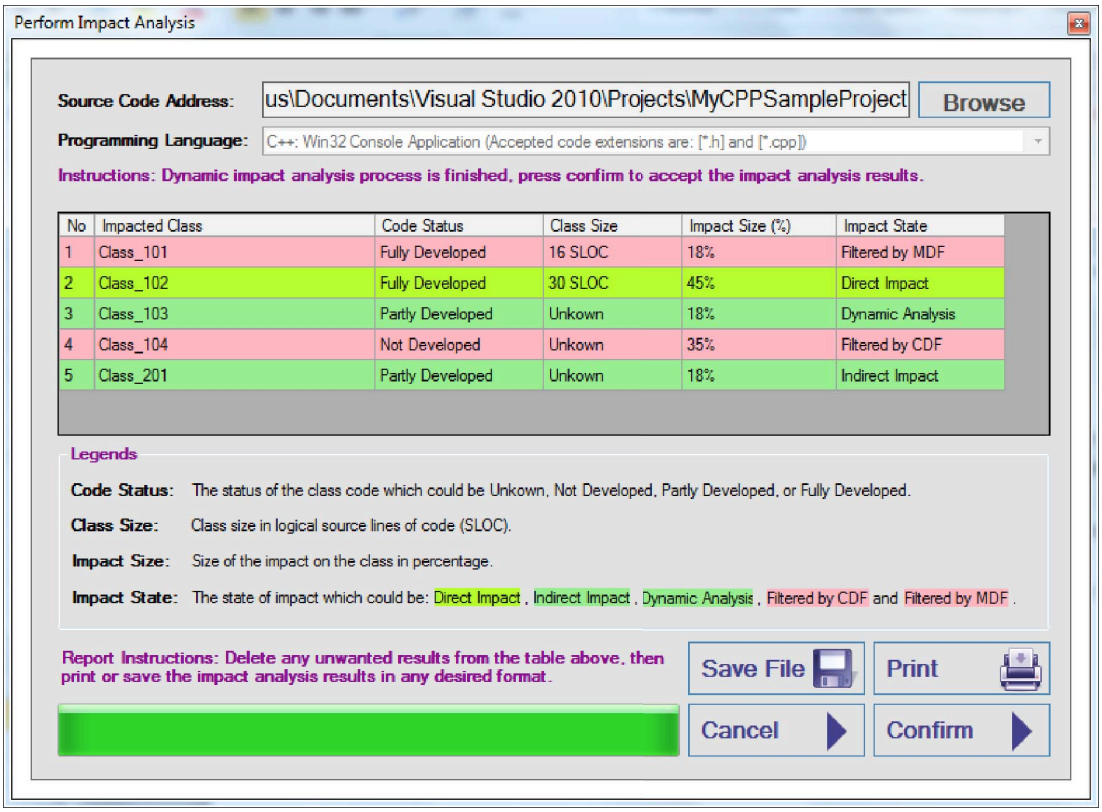

Fig. 7. A Sample of Change Impact Analysis Results in the Prototype Tool

our results, but it can be solved by calibrating the PV_{AT} in (1). Moreover, Fig. 6 shows the limited fluctuations of MRE values through the trend line, as well as the mean of the MRE values which is MMRE with a bold straight line. The MMRE value shown in Fig. 6 illuminates that the overall error rate of this approach is only 6.3%. This outstanding result justifies the accuracy of the CISE approach. However, in this evaluation the CISE approach has not been compared with other similar approaches for estimating change impact size, while their methods and goals were deviated with ours.

D. Threats to Validity

The evaluation procedure used in this paper also faces some threats to its validation. The environment used to implement the changes was very controlled and yet this approach might behave less accurate in other circumstance. This approach is not evaluated with incorrect CIP model; incorrect CIP model will definitely produce incorrect change impact analysis and change impact size estimation results. To sum up, further independent assessment of the approach in a different software projects and environments may generate more reliable results.

VI. Conclusion

This paper presented a new change impact size estimation (CISE) approach that utilizes an integrated impact analysis technique for the software development phase. This approach has two main functionalities which are analysing requirement changes and estimating the consequences of the requirement changes to the existing source codes. The results of this approach can be used for analysing the change request and predicting the consequences of change on the codes. The uniqueness of the CISE approach is because this approach is meant to estimate change impact size for the software development phase accurately.

Furthermore, a prototype tool has been developed to support this approach, which can automatically perform all the processes of integrated change impact analysis and change impact size estimation. Fig. 7 demonstrates a view of the prototype tool and a sample of it change impact size estimation results. Finally, this prototype tool has been used to evaluate the accuracy of CISE approach. Ten change requests in a case study has been used for evaluating the accuracy of the approach. The MMRE value which used to measure the mean error rate of this approach is calculated to be 6.30%, which demonstrates that the CISE approach is 93.7% 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 approach by considering more change requests from different case studies. Also, we will extend this approach for estimating change cost based on the approach results.

ACKNOWLEDGEMENT

The authors would like to thank the Lab of Advanced Informatics School for their offered helps, and all the members of the Lab for their useful discussions that guided us through this research. Also, to all academic staff and students of Advanced Informatics School who have been participating directly and indirectly in this study. The financial of this project is supported by Ministry of Higher Education Malaysia and Universiti Teknologi Malaysia under Vot No: 00K01.

REFERENCES

[1] I. Sommerville, *Software Engineering*. Addison-Wesley, 2007, ch. Configuration management, pp. 659–713.

[2] H. O. Ali, M. Z. A. Rozan, and A. M. Sharif, "Identifying challenges of change impact analysis for software projects," in *Innovation Management and Technology Research (ICIMTR), 2012 International Conference on*, pp. 407–411.

[3] D. Shao, S. Khurshid, and D. E. Perry, "Semantic impact and faults in source code changes: An empirical study," in *Software Engineering Conference, 2009. ASWEC '09. Australian*, pp. 131–141.

[4] H. M. Sneed and S. Huang, "Sizing maintenance tasks for web applications," in *Software Maintenance and Reengineering, 2007. CSMR '07. 11th European Conference on*, pp. 171–180.

[5] E. J. W. F. Murat Kahraman, GÄNGÖR; Ersin, "New change impact factor estimation in software development," *Turk. J. Elec. Eng. & Comp. Sci.*, vol. 20, pp. 1–14, 2012.

[6] N. Kama, T. French, and M. Reynolds, "Impact analysis using class interaction prediction approach," in *Proceedings of the 2010 conference on New Trends in Software Methodologies, Tools and Techniques: Proceedings of the 9th SoMeT_10*. 1860884: IOS Press, pp. 96–111.

[7] M. R. Nazri Kama, Tim French, *Considering Patterns in Class Interactions Prediction*, ser. Communications in Computer and Information Science. Springer Berlin Heidelberg, 2010, vol. 117, ch. 2, pp. 11–22.

[8] N. M. Kama, "A change impact analysis framework for the software development phase / mohd nazri kama," Thesis (Ph.D), 2011.

[9] M. H. A. Nazri Kama, "A change impact analysis approach for the software development phase: Evaluating an integration approach," *International Journal of Software Engineering and Its Applications*, 2012.

[10] S. Bohner, "Software change impacts-an evolving perspective," in *Proceedings of International Conference on Software Maintenance, 2002*, 2002, pp. 263 – 272.

[11] H. Lulu and S. Yeong-Tae, "Precise dynamic impact analysis with dependency analysis for object-oriented programs," in *Software Engineering Research, Management & Applications, 2007. SERA 2007. 5th ACIS International Conference on*, pp. 374–384.

[12] Y. Li, J. Li, Y. Yang, and M. Li, "Requirement-centric traceability for change impact analysis: a case study," in *Proceedings of the Software process, 2008 international conference on Making globally distributed software development a success story*. 1789770: Springer-Verlag, pp. 100–111.

[13] M.-A. Jashki, R. Zafarani, and E. Bagheri, "Towards a more efficient static software change impact analysis method," in *Proceedings of the 8th ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools and engineering*. 1512493: ACM, 2008, pp. 84–90.

[14] L. Huang and Y.-T. Song, "Precise dynamic impact analysis with dependency analysis for object-oriented programs," in *5th ACIS International Conference on Software Engineering Research, Management Applications, 2007. SERA 2007.*, aug. 2007, pp. 374 –384.

[15] B. Breech, M. Tegtmeyer, and L. Pollock, "Integrating influence mechanisms into impact analysis for increased precision," in *Software Maintenance, 2006. ICSM '06. 22nd IEEE International Conference on*, pp. 55–65.

[16] A. Rohatgi, A. Hamou-Lhadj, and J. Rilling, "An approach for mapping features to code based on static and dynamic analysis," in *Program Comprehension, 2008. ICPC 2008. The 16th IEEE International Conference on*, pp. 236–241.

[17] D. Leffingwell and D. Widrig, *Managing software requirements: a unified approach*. Addison-Wesley Longman Publishing Co., Inc., 2000.

[18] M. Singh and R. Vyas, "Requirements volatility in software development process," *International Journal of Soft Computing*, vol. 2, 2012.

[19] Z. D. Nurmuliani, N. and S. P. Williams, "Requirements volatility and its impact on change effort: Evidence-based research in software development projects," *11th Australian Workshop on Requirements Engineering, University of Adelaide SA*, 2006.

[20] N. Kama, T. French, and M. Reynolds, "Design patterns consideration in class interactions prediction development," *International Journal of Advance Science and Technology*, vol. vol. 28, pp. pp 45–64, 2011.

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

[22] J. Hihn and E. Monson, "Sizing the system : Jpl software estimation class for nasa," *Pasadena, CA : Jet Propulsion Laboratory, National Aeronautics and Space Administration, 2011.*, 2011.

[23] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empirical Software Engineering*, vol. 14, no. 2, pp. 131–164, 2009.

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

[25] V. Nguyen, B. Steece, and B. Boehm, "A constrained regression technique for cocomo calibration," pp. 213–222, 2008.