Using Static and Dynamic Impact Analysis for Effort Estimation

Nazri Kama, Sufyan Basri, Saiful Adli Ismail, and Roslina Ibrahim
Advanced Informatics School, Universiti Teknologi Malaysia, Malaysia

Abstract: *Effort estimation undoubtedly happens in both software maintenance and software development phases. Researchers have been inventing many techniques to estimate change effort prior to implementing the actual change and one of the techniques is using impact analysis. A challenge of estimating a change effort during developing a software is the management of inconsistent states of software artifacts i.e., partially completed and to be developed artifacts. Our paper presents a novel model for estimating a change effort during the software development phase through integration between static and dynamic impact analysis. Three case studies of software development projects have been selected to evaluate the effectiveness of the model using the Mean Magnitude of Relative Error (MMRE) and Percentage of Prediction (PRED) metrics. The results indicated that the model has 22% MMRE relative error on average and the accuracy of our prediction was more than 75% across all case studies.*

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

Received February 18, 2015; accepted September 26, 2016

1. Introduction

Changes happen across all stages in the software development phase. Calculating effort for a particular change request requires consideration on the status of the software artifacts i.e., partially completed, to be developed and completely developed. Many researchers have developed their own ways on the calculation strategies such as impact analysis strategy [1], expert judgment strategy [12], analogy strategy [20], function point analysis strategy [27], regression analysis strategy [7], and model-based strategy [2].

For impact analysis strategy [23], there are two types of techniques [14, 15] which are static and dynamic analysis techniques. On one hand, the static technique uses program static information (i.e., requirement, design, class and test artifacts) as an input for generating a set of potential impacted classes. On the other hand, the dynamic analysis technique uses program dynamic information or source code to develop a set of potential impacted classes.

In software development phase, estimating effort for a particular software change requires consideration on the inconsistency of software artifacts statuses. This is important because different statuses require different ways of estimation. In this paper, we propose a new change effort estimation technique that combines between static and dynamic analysis techniques [16]. The static analysis technique will be used to perform estimation on partially developed artifacts. This estimation is conducted on a set of potential impacted classes generated from high level documentation such as requirement document. For the dynamic analysis technique, it will be used for the completely developed artifacts. The estimation will be performed on a set of potential impacted classes that is generated from program runtime execution process.

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

2. Related Works

There are two main related keywords in this research which are impact analysis and effort estimation.

2.1. Impact Analysis

As described earlier, there are two impact analysis techniques[14, 15] which are the static analysis and the dynamic analysis techniques. On one hand for static analysis, there are two current static analysis techniques to our study which are Use Case Maps (UCM) technique [9] and the Class Interactions Prediction with Impact Prediction Filters (CIP-IPF) technique [17, 18].

The UCM technique [9] has two limitations which are:

1. There is no traceability technique used from the functional requirements and the high level design artifacts to the actual source codes. This technique only makes an assumption that the content of these two artifacts that is represented using the UCM model are reflected to the class artifacts.
2. There is no dynamic analysis or source code analysis involved in this technique.

Based on the precept that some of the effect of a change from a class to other class(es) may only be visible through dynamic or behavior analysis of the changed class [5, 19], results from this technique tend to miss some actual impacted classes. On the other hand, the CIP-IPF technique [17, 18] uses a class interactions prediction as a model for detecting impacted classes. This technique has its strength compared to the UCM technique. Comparing to the UCM technique, this technique has traceability link detection between the requirements artifacts and the class artifacts feature. This feature is used to navigate impact of changes at the requirement level to the class artifacts.

For the dynamic analysis techniques, we have selected two most related works to our research which are the Influence Mechanism technique [5] and the Path Impact technique [19]. Basically, these techniques predict the impact set (classes or methods) based on method level analysis. First, the Influence Mechanism technique [5] introduces the Influence Graph (IG) as a model to identify impacted classes. This technique uses the class artifacts as a source of analysis and assumes that the class artifacts are completely developed. There is a limitation for this technique which is there is no formal mapping or traceability process from requirements artifacts or design artifacts to class artifacts. This process is important in impact analysis process as changes not only come from class artifacts but it also comes from design and/or requirements artifacts. Since design and requirements artifacts do interact among them vertically (between two different artifacts of a same type) and horizontally (between requirement and design artifacts), changes that happen to them could contribute to different affected class artifacts. In some circumstances, focusing on the source code analysis may not able to detect those affected classes.

Next, the Path Impact technique [19] uses the Whole Path Directed Acyclic Graph (DAG) model as a model to identify impacted classes. The concept of implementation for this technique is almost similar to the Influence Mechanism technique as this technique uses the class artifacts as a source of analysis and assumes that the class artifacts are completely developed. Also, this technique performs a preliminary analysis prior to performing a detail analysis. There are two limitations of this technique. First, the implementation is time consuming as the technique opens to a huge number of data when the analysis goes to a large application. Next, there is no formal mapping process from requirements artifacts or design artifacts to class artifacts. As described earlier, this process is important in impact analysis process as changes not only come from class artifacts but also from design and/or requirements artifacts.

2.2. Effort Estimation

There are several categories of effort estimation which are:

1. Expert Judgment [12].
2. Estimation by Analogy [20].
3. Function Point Analysis [27].
4. Regression Analysis [7].
5. Model Based [2].

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 [20].

Traditionally, software size and effort are measured using Lines Of Code (LOC). However, earlier studies [27] showed that when the scale of the development grew, estimating using LOC failed to achieve accurate software effort estimation. Using different languages could also lead to 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 recommended 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 [2, 7]. Some multiple regression models also use other parameters such as development programming language or operating system as extra independent variables. The advantage of regression models is their mathematical basis as well as accuracy measurements.

3. The Approach

There are four steps in the approach which are:

1. Developing Class Interactions Prediction (CIP) model.
2. Acquiring change request attributes.
3. Performing change impact analysis.
4. Estimating required change effort.

3.1. Step 1: Developing Class Interactions Prediction Model

The CIP model is a model that shows traceability relationships among all software artifacts (requirements, designs and classes). This model will be used as a static model in which the effort estimation will be conducted based on this model. Further explanation on the development of the model can be referred to [14, 15].

3.2. Step 2: Acquiring Change Request Attributes

This step acquires change request attributes which has direct impact on the effort estimation results. According to one of the previous works [22], one of the important attribute is type of change.

3.3. Step 3: Performing Change Impact Analysis

The Change Impact Analysis step consists of two stages:

1. Static analysis.
2. Dynamic analysis.

In the first stage, static impact analysis is performed on the established CIP model to identify the impacted classes i.e., direct and indirect. Initially, the static impact analysis identifies the first layer of the class artifacts that are affected by the requirement or software changes. These class artifacts are identified as the direct impacted classes. In this stage, vertical traceability relations are not considered first. Then the static impact analysis continues with the second and onward levels of the class artifacts from the CIP model. These class artifacts, on the other hand, are identified as the indirect impacted classes.

The static impact analysis process uses a Breadth First Search (BFS) technique on the CIP model [28] to identify the impacted class artifacts. The technique defines the impacted class artifacts as the search process objective and each software artifact as the node of the search path.

The static impact analysis process continues by conducting a further refinement on the static results, to eliminate the incorrectly expected results due to excessive prediction. The technique used for the refinement is Impacted Class Purification (ICP). The ICP process eliminates the incorrectly impacted class artifacts using the traceability among classes dependencies. This tracing process also known as detection process is a common impact analysis process that has been automated by several researchers [6, 11]. The tracing starts from the indirectly impacted class artifacts to any of the direct impacted class artifacts. The impacted class artifact is removed from the result if no valid traceability exists. The traceability mapping is conducted on the CIP model using the vertical and horizontal software artifacts dependencies. The output of the ICP process produces the final result of the static impact analysis process as in Figure 1 that is considered as the input for the next stage.

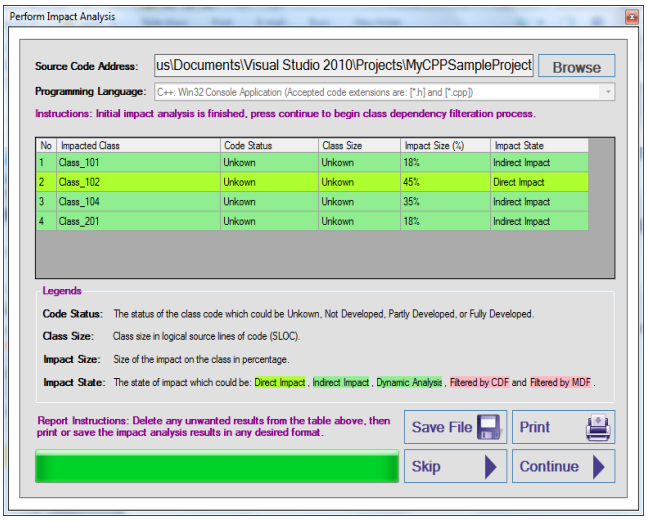

Figure 1. Sample of static impact analysis results.

In the next stage, the dynamic analysis starts with code status detection and traceability update using pattern detection. The code status detection is required as part of the consideration of the existence of the not developed, partially developed and fully developed classes during software development phase. The fully developed class detection is important for the next step, Method Execution Path creation. Next, our approach further enhances the current dynamic analysis technique by intervening the traceability pattern detection to further improve the impacted classes information.

For the code status detection, three types of class artifacts are identified:

1. Not developed.
2. Partially developed.
3. Fully developed.

The class is considered as not developed if a class exists without any declaration or if there is no concrete function implementation in the code files. To avoid ambiguity, further marking technique is introduced using a special tagging for each code file to maintain the code status.

The construction of the marking technique for the code status special tagging is described as: [special tagging + "<status>" +code status+"</status>"], where special tagging is subjected to the programming

language, and code status defines the possible status of the codes:

1. Not Developed.
2. Partially Developed.
3. Fully Developed.

There are also other possible code statuses in a typical software development implementation such as "Stubbed" and "Faked".

Stubbed procedure is a test function that is used to link and verify other codes or classes. Faked procedure is a bogus function that looks real and workable function, but it returns a fixed response without specific purpose. However, these code statuses are not significant for our approach consideration and hence are ignored.

Subsequently, the approach introduces traceability patterns to handle the traceability issues especially in the Agile methodology. The Traditional methodology traceability approach has been restricted in the Agile methodology as requirements are captured and communicated through an informal approach. There is limited evidence in the Agile methodology software development that implement detailed design which relate the requirement to class artifacts, thus constraint the Traditional methodology traceability approach. Ghazarian proposed a similar approach to our code status detection approach, which is using special tag in the class artifacts [8].

We have further improved the technique [8] to specify the requirement-class interaction as follow: [Special comments mark+"<requirement>"+ Requirement Traceability + "</requirement>"], where special comments mark depends on the programming language used, and the Requirement Traceability identifies the requirement ID and description or product backlog in the Agile methodology software development. Although the evolution of source code and requirement and constrained with the traceability patterns in each iteration, however the source code and requirement are traceable to each other. As a result, the approach could produce a more accurate refined set of the impacted classes.

Once the filtered set of impacted classes are obtained, the method execution paths are created from fully developed classes. The actual interaction between the classes can be determined from the created method executions paths. Afterwards, the CIP model is updated with the actual class interactions. Finally, the Method Dependency Filtration (MDF) process is performed similar to ICP process on the impacted classes to filter the overestimated impact analysis results. Figure 2 shows the sample of dynamic impact analysis results.

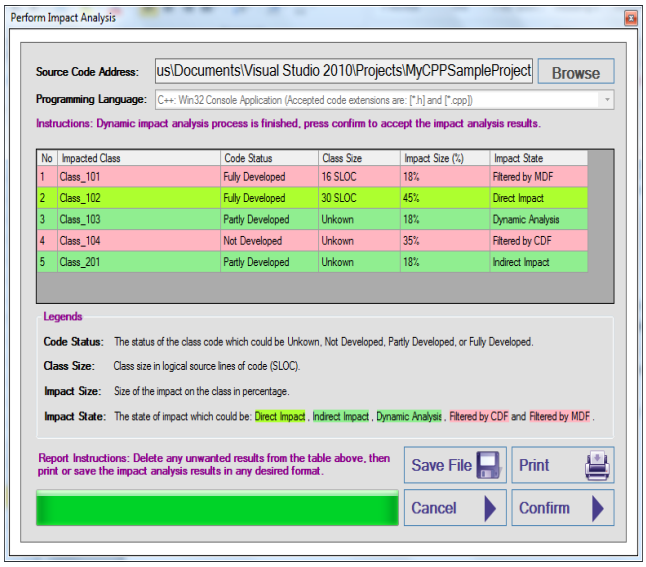

Figure 2. Sample of dynamic impact analysis results.

The improved filtered set of impacted classes using this process is the final impact analysis result in the method. This sequence of methods implies that by having fully developed classes, an accurate impact analysis can be performed, even with inaccurate CIP model from the beginning, which is very crucial from the software development perspectives. Finally, based on the final filtered set of impacted classes, the prediction of the potential change impact size of each impacted class is calculated.

3.4. Step 4: Estimating Required Change Effort

The last step estimates the required change effort based on the initial effort estimation and the combination of static and dynamic impact analysis results. To estimate the change effort based on COCOMO 2 effort estimation [24], we introduce a mathematical equation to calculate change effort *CPM* according to the original estimated effort *PM* and updated effort estimation *PM'* as Equation (1). CPM is the total effort need to implement the change; it is equal to priority multiplier multiplied by the deviation of estimated effort with new software size *PM'* and original estimated effort *PM* plus the extra effort needed to change the developed code as the follow:

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

Where *DSF* is the development status factor based on Equation (7), *PM* is the original estimated effort using COCOMO II in man per month, *PM'* is the updated estimated effort after change using new software size in man per month and it is calculated using Equation (2) and *PR* is the priority multiplier which is determined by the effect of the change request priority and how much it will affect the change effort; this value should be selected according to the development methodology of the development group.

Equations (2), (3), and (4) below shows how *PM′* is calculated. This equation will be justified with the assumption that the cost factors and the scale factors [22] will not change with the change request. Accordingly, the mathematical justification for producing this Equation is as follow:

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

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

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

Where *PM* is the original estimated effort using COCOMO 2 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 Equation (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* stands for scale factor, which will be derived from the five scale factors.

Assuming that the initial effort estimation was done before the change request, the only unknown variable in Equation (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 Equation (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 presented in our previous paper in the static impact analysis steps [4].

DSF in Equation (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, more effort will be needed 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 [24]. *DSF* will be calculated using the following Equation (7):

$$DSF = \left(\frac{(ND \times NND) + (PD \times NPD) + (FD \times NFD) - NIC}{NIC} \right) \tag{7}$$

where DSF stands for Development Status Factor (DSF ≥ 0), *ND* is equal to affected multiplier for not developed classes, *NND* is the number of not developed impacted classes, *PD* is equal to affected multiplier for partly developed classes, *NPD* is the number of partly developed impacted classes, *FD* is equal to affected multiplier for fully developed classes, *NFD* is the number of fully developed impacted classes, *NIC* is the total number of impacted classes.

The 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[26] which could be used to estimate multiplier values as described in our previous paper [4].

In this research, our approach is developed for Early Design sub-model of COCOMO 2 [25] which uses SLOC as the software size metric. Therefore, we use logical SLOC as the code size; however, this model can easily be adapted for other COCOMO 2 sub-models [25] and can also use Function Points as software size metric.

4. Evaluation

This section describes the evaluation of our approach.

4.1. Case Study

To measure the accuracy of the approach, we have implemented the approach in three case studies of software projects which implemented different type of software development process (see Table 1).

Table 1. Case studies.

Case study	Project Name	Software Development Process
CS1	Centralized Access Control	Agile Unified Process (AUP)
CS2	User Management and Verification System	Scrum
CS3	Password Management System	Extreme Programming (XP)

4.2. Data Collection

From three Case Studies (CS) with different software development process and the change types, 73 change requests have been recorded, and the distribution of the change requests is presented in Table 2.

Table 2. Change Requests per Case Study.

Case study	Number of ChangeRequests
CS1	27
CS2	25
CS3	21

4.3. Evaluation Metrics

For evaluating the accuracy of the approach, three effort estimation metrics have been used which are Magnitude of Relative Error (MRE) [13], Mean Magnitude of Relative Error (MMRE) [21], and Percentage of Prediction, *PRED* (.25) [10].

MRE: a metric for the absolute estimation accuracy only[13]. This metric calculates a rate of the relative errors in both cases of over-estimation or under-estimation as shown in Equation (8).

$$MRE = abs\left[\frac{Actual\ \text{Results} - Estimated\ \text{Results}}{Actual\ \text{Results}}\right] \quad (8)$$

MMRE: Mean Magnitude of Relative Error is the percentage of average of the MREs over an entire data set [21]. It is used for calculating the accuracy of an estimation technique using T number of tests as it is shown in Equation (9).

$$MMRE = \frac{100}{t}\sum_{i}^{t} MRE_i \quad (9)$$

The *MRE* metric will be calculated for each predicted impacted class from the change request experience to measure the accuracy of the change effort estimation in our approach. But the *MMRE* will be calculated for the whole case study, which contains 73 change requests. The results of our approach are more accurate when the *MMRE* values are smaller.

Percentage of prediction, *PRED* (.25) is the percentage of estimates that falls within 25 percent of the actual value [10]. Percentage of prediction definition is shown in Equation (10), where *K* is the number of estimations where MRE value is less or equal to x and n is the total number of estimations.

$$PRED(x) = \frac{K}{n} \quad (10)$$

4.4. Evaluation Procedure

There are three main steps in the evaluation which are:

1. Estimating change effort results using the new approach.
2. Gathering the actual change implementation effort from the project reports.
3. Comparing results between the estimated change effort with the actual change effort.

5. Result and Discussion

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

According to [3] 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 [10], proposed that an acceptable *MMRE* value (or error rate) for software effort estimation is 25%. This value shows that on average, the accuracy of the estimation is more than 75%. For our evaluation, we have used this guideline to assess the accuracy of our proposed approach by targeting the *MMRE* value (or acceptable error rate) should be less than 25%. We also used *PRED* (.25) as the second evaluation metric to support the result produced by *MMRE*.

Since our 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 3 shows the *MRE*, *MMRE* and *PRED* (.25) of change requests in each case study.

Table 3. MMRE, Overall MMRE and PRED (.25) based on Change Requests (CT) across Case Study (CS).

Case study	MMRE (%)	Overall MMRE (%)	PRED (.25)
CS1	22%	22%	77%
CS2	24%		
CS3	20%		

A quick look on the average *MMRE* value revealed that:

- Our model has 22% relative error on average which is better than our expectation.
- All *MMRE* values for the case studies is less than 25%.
- The percentage of prediction, *PRED* (.25) revealed that the accuracy of our approach is more than 75% for all case studies.

This preliminary analysis indicated that the proposed approach of change effort estimation model is acceptably accurate. However, the accuracy results need to be further investigated and analyzed.

6. Conclusions

We have developed a new approach that estimates change effort for a particular change request during software development phase. The novelty of this paper resides in the estimation of a change effort during software development phase through integration between static and dynamic impact analysis. Three case studies have been selected to evaluate the effectiveness of the model using the (MMRE) and Percentage of Prediction (PRED) metrics. The results indicated that the model has 22% MMRE relative error on average and the accuracy of our prediction is more than 75% across all case studies.

References

[1] Asl M. and Kama N., "A Change Impact Size Estimation Approach During the Software Development," *in Proceedings of 22nd Australian Software Engineering Conference*, Melbourne, pp. 68-77, 2013.

[2] Attarzadeh I., Mehranzadeh A., and Barati A., "Proposing an Enhanced Artificial Neural Network Prediction Model to Improve the Accuracy in Software Effort Estimation," *in Proceedings of 4th International Conference on Computational Intelligence, Communication Systems and Networks*, Phuket, pp. 167-172, 2012.

[3] Basha S. and Ponnurangam D., "Analysis of Empirical Software Effort Estimation Models," *Software Engineering*, vol. 7, no. 3, 2010.

[4] Basri S., Kama N., Adli S., and Haneem F., "Using Static and Dynamic Impact Analysis for Effort Estimation," *IET Software*, vol. 10, no. 4, pp. 89-95, 2016.

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

[6] Fasolino A. and Visaggio G., "Improving Software Comprehension Through An Automated Dependency Tracer," *in Proceedings 17th International Workshop on Program Comprehension*, Pittsburgh, pp. 58-65, 1999.

[7] Garcia L., Augusto C., and Hirata C., "Integrating Functional Metrics, COCOMO II and Earned Value Analysis for Software Projects Using PMBoK," *in Proceedings of the ACM Symposium on Applied Computing*, Fortaleza, pp. 820-825, 2008.

[8] Ghazarian A., "Traceability Patterns: An Approach to Requirement-Component Traceability in Agile Software Development," *in Proceedings of the 8th Conference on Applied Computer scince World Scientific and Engineering Academy and Society*, Venice, pp. 236-241, 2008.

[9] Hassine J., Rilling J., Hewitt J., and Dssouli R., "Change Impact Analysis for Requirement Evolution Using use Case Maps," *in Proceedings of 8th International Workshop on Principles of Software Evolution*, Lisbon, pp. 81-90, 2005.

[10] Huang S., Chiu N., and Chen L., "Integration of the Grey Relational Analysis with Genetic Algorithm for Software Effort Estimation," *European Journal of Operational Research*, vol. 188, no. 3, pp. 898-909, 2008.

[11] Ibrahim S., Idris N., Munro M., and Deraman A., "Integrating Software Traceability for Change Impact Analysis," *The Arab International Journal of International Technology*, vol. 2, no. 4, pp. 301-308, 2005.

[12] Jorgensen M., "Practical Guidelines for Expert-Judgment-Based Software Effort Estimation," *IEEE Software*, vol. 22, no. 3, pp. 57-63, 2005.

[13] Jorgensen M. and Molokken-Ostvold K., "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.

[14] Kama N., "Change Impact Analysis for the Software Development Phase: State-of-the-Art," *International Journal of Software Engineering and Its Applications*, vol. 7, no. 2, pp. 235-244, 2013.

[15] Kama N., "Integrated Change Impact Analysis Approach for the Software Development Phase," *International Journal of Software Engineering and its Applications*, vol. 7, no. 2, pp. 293-304, 2013.

[16] Kama N., French T., and Reynolds M., "Impact Analysis using Class Interaction Prediction Approach," *in Proceedings of Conference on New Trends in Software Methodologies, Tools and Techniques: Proceedings of the 9th SoMeT_10*, Amsterdam, pp. 96-111, 2010.

[17] Kama N., French T., and Reynolds M., "Predicting Class Interactions from Requirement Interactions: Evaluating a New Filtration Approach," *in Proceedings of Software Engineering*, Innsbruck, 2010.

[18] Kama N. and Ridzab F., "Requirement Level Impact Analysis with Impact Prediction Filter," *in Proceedings of International Conference on Software Technology and Engineering*, Phuket, 2012.

[19] Law J. and Rothermel G., "Whole Program Path-Based Dynamic Impact Analysis," *in Proceedings of 25th International Conference on Software Engineering*, Portland, pp. 308-3018, 2003.

[20] Li J., Ruhe G., Al-Emran A., and Richter M., "A Flexible Method for Software Effort Estimation by Analogy," *Empirical Software Engineering*, vol. 12, no. 1, pp. 65-106, 2007.

[21] Nguyen V., Steece B., and Boehm B., "A Constrained Regression Technique For Cocomo Calibration," *in Proceedings of the 2nd ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, Kaiserslautern, pp. 213-222, 2008.

[22] Nurmuliani N., Zowghi D., and Williams S., "Requirements Volatility and its Impact on Change Effort: Evidence-Based Research in Software Development Projects," *in Proceedings*

of the 11th Australian Workshop on Requirements Engineering, Adelaide, 2006.

[23] Pfleeger S. and Bohner S., "A Framework for Software Maintenance Metrics," *in Proceedings of Conference on Software Maintenance*, San Diego, pp. 320-327, 1990.

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

[25] Yang D., Wan Y., Tang Z., Wu S., He M., and Li M., "COCOMO-U: An Extension of COCOMO II for Cost Estimation with Uncertainty," *in Proceedings of Software Process Change*, Shanghai, pp. 132-141, 2006.

[26] Yang Y., He M., Li M., Wang Q., and Boehm B., "Phase Distribution of Software Development Effort," *in Proceedings of the 2nd ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, Kaiserslautern, pp. 61-69, 2008.

[27] Zheng Y., Wang B., Zheng Y., and Shi Li., "Estimation of Software Projects Effort based on Function Point," *in Proceedings of 4th International Conference on Computer Science and Education*, Nanning, pp. 941-943, 2009.

[28] Zhou R. and Hansen E., "Breadth-First Heuristic Search," *Artificial Intelligence*, vol. 170, no. 4-5, p. 385-408, 2006.

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

Sufyan Basri obtained his first degree at Universiti Teknologi Malaysia (UTM) in Electrical Engineering (Mechatronic) in 2001, master degree in Real Time Software Engineering in 2003 and PhD in Software Engineering in 2016 at the same university. His research interest includes Software Engineering, Change Management and Effort Estimation. He has more than 10 years of experiences for various software development projects and implementations.

Saiful Adli Ismail obtained his first degree at Universiti Teknologi Malaysia (UTM) in Industrial Computing in 1997, second degree in Real Time Software Engineering at the same university in 2000 and currently continues his PhD at Universiti Teknologi Malaysia.

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