Considering Partially Developed Artifacts in Change Impact Analysis Implementation

Nazri Kama and Sufyan Basri
Universiti Teknologi Malaysia, Kuala Lumpur, Malaysia
Email: nazrikama@ic.utm.my, msufyan4@live.utm.my

***Abstract*—It is important to manage the changes in the software to meet the evolving needs of the customer and hence, satisfy them. Accepting too many changes causes delay in the completion and it incurs additional cost. One type of information that helps to make the decision is the prediction of the number of classes affected by the changes or change impact analysis. Current impact analysis approaches assume that all classes in the class artifact are completely developed and the class artifact is used as a source of analysis since it represents the final user requirements. However, these assumptions are impractical for impact analysis in the software development phase as some classes in the class artifact are still under development or partially developed that leads to inaccuracy. This paper presents a novel impact analysis approach to be used in the software development phase. The significant achievements of the approach are demonstrated through an extensive experimental validation using several case studies. The experimental analysis shows improvement in the accuracy over current impact analysis results.**

***Index Terms*—Impact analysis, class interaction, requirement interaction, traceability, software development**

I. Introduction

Current impact analysis techniques include static analysis techniques [1] [2] [3] [4] and dynamic analysis techniques [5] [6] [7] [8]. These techniques are mainly developed for the software maintenance phase. The implementation of these techniques is based on the assumptions that: (1) all classes in the class artifacts are completely developed; and (2) the class artifact is used as a source of analysis since it represents the final forms of user requirements [9]. Unfortunately, these assumptions are not practical for implementing impact analysis in the software development phase since some classes in the class artifacts are still under development or partially developed [10].

The existence of partially developed classes in the class artifacts causes several problems to these static analysis and dynamic analysis techniques. The static analysis technique faces a problem related to the accuracy of program static information (i.e., class interactions) that is generated from source code through reverse engineering. The generated class interactions that involve partially developed classes may not represent the actual class interactions as some of the interactions have not been developed yet. On the other hand the dynamic analysis techniques tend to produce inaccurate method execution paths that are generated from source code through reverse engineering. This is because some method execution paths that involve partially developed classes may have not been developed yet. The inaccuracy of the generated program static information from the static analysis technique and method execution paths from the dynamic analysis technique indirectly lead to inaccuracy of impact analysis results.

We propose a new approach to perform impact analysis during software development. Our approach combines current static and dynamic analysis techniques, and supplements actual class interactions derived from source code with inferred class interactions derived from the requirements.

This paper is laid out as follows: Section 2 justifies past related works. Next, Section 3 describes the new impact analysis approach. Thereafter, Section 4 and Section 5 present evaluation strategy and results. Finally, conclusion and future work are explained Section 6 and Section 7.

II. Related Work

One of the most referred definitions of impact analysis is a process of identifying potential consequences of a change, or estimating what needs to be modified to accomplish a change [11].

There are two main perspectives to impact analysis which are the dependency analysis and the traceability analysis. Typically, the dependency analysis is also known as a program analysis. The program analysis focuses on identifying relationships among class artifacts or source codes by exploring the internal structure of the codes [11]. This analysis aims to determine what elements in the source codes could be potentially affected by a change. There are many types of program analysis techniques that have been introduced, such as the control dependency and the data dependency [12]. The control dependency uses a program's conditional structures for the analysis whereas the data dependency analyses the program's variable.

Comparatively to the program analysis, the traceability analysis is the analysis of relationships between software artifacts across different software phases. Since this

Manuscript received October 7, 2013; revised December 2, 2013; accepted January 8, 2014.

doi:10.4304/jsw.9.8.2174-2179

analysis involves various software artifacts across different software phases, some researchers use this analysis to support impact analysis activity for the software development phase [13] [14]. The difference between this analysis and the program analysis is that this analysis focuses on the dependencies between software artifacts in different software phases instead of a single software artifact. There are two types of traceability analysis which are the Pre-traceability analysis and the Post-traceability analysis [15]. The pre-requirement traceability provides a mechanism to verify that all requirements have been described in a formal requirement specification document. On the other hand, the post-requirement traceability provides a mechanism to ensure all requirements in the formal requirement specification document have been implemented and how they have been implemented in the software.

Much of the work on impact analysis has been limited to source code analysis [5] [8] [11] using the dependency analysis approach. Relying on the source code analysis does not account for the overall impact to a software project [1] [16]. Software artifacts such as design and test artifacts should be kept up-to-date according to the change. This indirectly shows that these software artifacts are part of the impacted artifacts by the change. Thus, to identify thorough consequences of making a change in a software project, an effective combination between the traceability analysis and the dependency analysis is important.

III. A New Change Impact Analysis Approach

This section describes an overall structure of a new impact analysis approach for the software development phase (will be called the Software Development Phase Change Impact Analysis (SDP-CIA)). The new approach is a direct extension of the Class Interaction Prediction with Impact Prediction Filters (CIP-IPF) technique [14]. The difference between the CIP-IPF technique [14] and this approach is the inclusion of the dynamic analysis technique in the impact analysis process implementation. In brief, there are two main stages in the approach which are: (1) Stage 1- Developing the program static information (i.e., class interactions prediction) and; (2) Stage 2- Performing impact analysis.

A. Stage 1: Developing Class Interactions Prediction

This approach uses a predictive technique to develop class interactions prediction model. In brief, the new predictive technique develops the class interactions prediction based on two analyses which are: (1) significant object interactions analysis in the requirement artifact; (2) design patterns analysis in the design artifact. The first analysis analyses the significant object interactions in the requirement artifact to develop an initial class interactions prediction via horizontal traceability links. For the horizontal traceability links, the new predictive technique refines the selected current technique which is the Rule-based technique [17] [18].

The second analysis is the design patterns analysis. This analysis is considered as an important analysis for the new predictive technique as the current techniques [19] [20] [21] do not exploit the design artifacts. This analysis modifies the initial class interactions prediction produced by the first analysis according to design patterns. At this moment, this stage performs the analysis according to the Boundary-Controller-Entity (BCE) design pattern only. However, the developed steps for this analysis are flexible in that it can also be used to implement other design patterns analyses.

There are four processes in this stage which are: (1) Extracting software artifact elements process; (2) Detecting traceability link process; (3) Developing initial class interactions prediction process; and (4) Modifying the initial class interactions prediction.

B. Stage 2: Performing Impact Analysis

This stage identifies a set of potential impacted classes using the class interactions prediction (Stage 1 result) according to change requests. There are two main processes in this stage which are the impact analysis process and filtration process.

Impact Analysis Process: There are three steps in this process which are: Step 1: Identifying a set of impacted requirements; Step 2: Identifying a set of impacted design classes; and Step 3: Identifying an initial set of potential impacted classes. The outcome of this process is an initial set of impacted classes that will be used by the next process (the filtration process) to filter false impacted classes in the initial set of impacted classes if they exist.

Filtration Process: This process eliminates some typically false results generated by the impact analysis process. There are two filtration levels in this process which are the Class Dependency Filtration (CDF) level and the Method Dependency Filtration (MDF) level.

Method 1 in Class A Pseudocode:

```
BEGIN (P1)
    Temperature = ThermometerRead(41)
    IF Temperature > 40 THEN
        PRINT "it's HOT!"
    END IF
END (P1)
```

Method 2 in Class B Pseudocode:

```
BEGIN ThermometerRead(Source
insideOrOutside)
    // to be developed
    RETURN
END ThermometerRead
```

Figure 1. Example of stub

The CDF level implements the static analysis on the initial set of potential impacted classes produced by the impact analysis process. This implementation is important by the fact that some interaction links in the initial set of potential impacted classes have no change impact value. The interaction link that has no change impact value means that if change happens to one side of two interacting classes, the other class will not be affected. This is because the other class does not require the changed class for its implementation.

The MDF level performs another filtration on the filtered set of potential impacted classes produced by the CDF level. In brief, all method execution paths from the filtered set of potential impacted classes will be extracted and further analysed to eliminate false impacted classes. We use the backward and forward analysis technique from [5] for the elimination technique. This level can be considered as the dynamic analysis level as it uses the

method execution paths to identify potential impacted classes.

Generally, most dynamic analysis techniques [6] [7] [22] [23] consist of two main steps in its implementation. These steps are extracting method execution paths from the application and analysing the generated method execution paths to identify a set of potential impacted classes (according to change request). The first step focuses on generating method execution paths from completed classes using a path generator tool. There are many existing path generator tools that can be used for the generation such as Code Surfer [24] or IBM Rational Application Developer [25]. The next step analyses the generated method execution paths to detect a set of actual impacted classes according to change requests using detection technique such as the backward or forward technique from [5], the global tracking-based algorithm and the influence graph-based algorithm from [8].

There are two main disadvantages of the current dynamic impact analysis techniques from the software development phase perspective which are: (1) all techniques are developed to support change impact analysis in the software maintenance phase, and; (2) all techniques do not consider or include partially developed class analysis in its implementation. This second disadvantage occurs because of all classes in the software maintenance phase have been completely developed or fully developed. Thus, it is not important for these techniques to have the partially developed class analysis.

From the dynamic impact analysis implementation in the software development phase, the inclusion of partially developed class analysis is an important feature. This is due to in the software development phase where situation of some classes in the class artifacts are still under construction or partially developed exist. This inclusion is required to ensure the accuracy of the extracted or generated method execution paths from class artifacts. This accuracy indirectly contributes to the accuracy of the set of potential impacted classes results.

Prior to demonstrating the importance of the inclusion of partially developed class analysis in the dynamic analysis technique, the partially developed class is defined as a class that consists of some undeveloped methods. Typically, this undeveloped method is replaced using a dummy code or a stub [26]. Fig. 2 is an example of stub.

```
M0()
{
    M1();
    IF Cond1 THEN M2();
        ELSE M3();
    WHILE Cond2 DO
    {
        M4();
        M5();
    }
    M6();
}

M2()
{
    M8(); //method stub
    M10();
}
M3 ()
{
    M8();//method stub
}

M6()
{
    IF Cond4 THEN
M7();
        ELSE M9();
}
M8 ()*- Partially
Developed
{
}
```

Figure 2. Example of several methods' algorithms

Fig. 2 shows a Method 1 in Class A reads a temperature using a ThermometerRead function call from Method 2 in Class B. Since the Method 2 has yet to be completely developed, the ThermometerRead function is replaced using a default value (41) to represent the Method 2 functionality. In this case, the "ThermometerRead(41)" is considered as the stub. To demonstrate the importance of the inclusion of partially developed class analysis in the software development phase.

Assuming that M8 is partially developed, M2 and M3 consist of M8 stubs. This stub will not call actual M8 method (see asterisk (*) symbol represents the actual M8 implementation). Based on the path generator tool (IBM Rational Application Developer tool [25]), among the extracted method execution paths are: (1) Path 1: M, M1, M2, M10, M4, M5, M6, M7 and; (2) Path 2: M, M1, M3, M4, M5, M6, M7. However based on the actual method execution paths, the actual path (based on Path 1 and Path 2) are: (1) Path 1: M, M1, M2, M8, M10, M4, M5, M6, M7 and; (2) Path 2: M, M1, M3, M8, M4, M5, M6, M7.

The difference between the generated method execution paths using automated tool and the actual method execution paths can be seen after M2 execution. The path generator tool does not consider M8 after M2 in the generated paths as the statement to call M8 is replaced by a stub. Therefore, by looking at this example, there are two important aspects can be seen. First, the stub has caused the method execution paths generated are not reflected to the actual method execution paths. Second, the partially developed class analysis/stub analysis consideration is important to produce an accurate potential set of method execution paths.

IV. Evaluation

This section describes the evaluation strategy that is established to measure the effectiveness of the new proposed approach (will be called the "SDP-CIA" Software Development Phase Change Impact Analysis). Basically the measurement will answer a question of "Does the SDP-CIA give better accuracy of impact analysis results than the selected current impact analysis approaches?" The selected current impact analysis approaches are: (1) the Class Interactions Prediction with Impact Prediction Filters (CIP-IPF) approach [27] [28] and (2) Path Impact approach [5].

A. Subject and Case Study

The subjects of the experiment were three groups of final year post-graduate students of software engineering course at Advanced Informatics School, Universiti Teknologi Malaysia (UTM). During their professional attachment session in the industry, we were involved as one of the software developers in these projects. We developed some of the modules which were then used as the case study.

For the purpose of performing the impact analysis evaluation, the author issues a set of change requests to the developed modules and the impact analysis results according to the issued change requests are then identified. This experiment requires the subjects to use or implement three different impact analysis approaches which are the CIP-IPF approach [27] [28], the Path-Impact approach [5] and the SDP-CIA. The subjects were given a preliminary guideline and briefing on these

techniques prior to the experiment. The guideline includes thorough technique explanations and example of its implementation.

B. Evaluation Metrics

This study employed evaluation metrics as described in [29] [30]. The reason why we choose this metrics is that this metric has been used by several researchers [1] [5] [10] to measure the effectiveness of impact analysis prediction. Briefly, each impacted class predictions were categorized according to four numbers:

- Not Predicting and Not Changing (NP-NC): number of pairs of classes correctly predicted to not be changing
- Predicting and Not Changing (P-NC): number of pairs incorrectly predicted to be changing.
- Not Predicting and Changing (NP-C): number of classes incorrectly predicted to not be changing.
- Predicting and Changing (P-C): number of classes correctly predicted to be changing.

These numbers are then used to calculate:

- Completeness value: The ratio of the actual class interactions or impacted classes that were predicted.
- Correctness value: The ratio of the predicted class interactions that were actually interacting or impacted classes that were actually impacted.
- Kappa value: This value reflects the accuracy or the prediction (0 is no better than random chance, 0.4-0.6 is moderate agreement, 0.6-0.8 is substantial agreement, and 0.8-1 is almost perfect agreement [31].

C. Hypotheses

A hypothesis that investigates the effectiveness of the new approach (to recap, we call it SDP-CIA) is developed. The SDP-CIA represents a combination of the static and dynamic analysis approaches whereby the selected current impact analysis approaches represent the independent technique (CIP-IPF- static analysis approach only; Path Impact- dynamic analysis approach only). If the combination is not effective, H_0 is accepted. Otherwise, H_0 is rejected. The hypothesis is:

- H_0: The SDP-CIAF does not give higher accuracy of impact analysis results than the selected current techniques results
- H_a: The SDP-CIAF gives higher accuracy of impact analysis results than the selected current techniques results

V. Evaluation Results

Table I shows impact analysis results produced by all impact analysis approaches (the CIP-IPF, the Path Impact technique and the SDP-CIA).

To validate the hypothesis, the Independent T-Test statistical analysis is used. Two stages of analysis are created. The first stage compares Means results between the CIP-IPF approach and the SDP-CIA approach whereas the second stage compares Means results between the Path Impact technique and the SDP-CIA approach.

TABLE I.
INDEPENDENT T-TEST RESULTS BETWEEN THE CIP-IPF AND THE SDP-CIA

The Technique	Means Results
CIP-IPF Technique	0.7927
SDP-CIA	0.9060

A. Stage 1 Analysis: The CIP-IPF Technique vs. The SDP-CIA

Table II shows the Independent T-Test results.

TABLE II.
IMPACT ANALYSIS RESULTS

CRID	CIP-IPF			Path Impact			SDP-CIA		
	Com (%)	Corr (%)	Kappa Value	Com (%)	Corr (%)	Kappa Value	Com (%)	Corr (%)	Kappa Value
CR1	80	100	0.785	66.7	100	0.652	86.7	100	0.876
CR2	81.3	100	0.821	78.6	100	0.789	92.9	100	0.935
CR3	76.9	100	0.768	80	92.3	0.752	100	93.8	0.944
CR4	83	94	0.795	88.7	94.1	0.85	94.4	94.4	0.903
CR5	83	91	0.767	91.7	91.7	0.852	91.7	91.7	0.852
CR6	82.4	100	0.832	76.5	92.9	0.721	94.1	94.1	0.842
CR7	81.8	90	0.734	80	94.1	0.764	95	95	0.912
CR8	80	100	0.806	78.6	100	0.787	92.9	100	0.935
CR9	75	100	0.752	87.5	100	0.884	87.5	100	0.884
CR10	76	100	0.77	88.2	100	0.892	94.1	100	0.947
CR11	85.7	100	0.863	73.7	93.3	0.695	94.7	94.7	0.908
CR12	80	100	0.773	68.8	100	0.676	87.5	100	0.884
CR13	90.9	90.9	0.83	76.5	100	0.769	94.1	100	0.947
CR14	83	100	0.843	77.8	100	0.784	94.4	100	0.95
CR15	80	92	0.749	80	100	0.804	86.7	100	0.874

The null hypothesis for the Independent T-Test was that the Kappa mean value from both approaches are equal, H_0: $\mu 1=\mu 2$. The null hypothesis is accepted if the Sig. (2-tailed) value is greater than 0.05. The alternative hypothesis was used to reject the null hypothesis if the Kappa means values from both approaches are not equal, H_0: $\mu 1\neq\mu 2$. The alternative hypothesis is accepted if the Sig. (2-tailed) value is less than 0.05.

To answer the question of "Does the SDP-CIAF give better accuracy of impact analysis results than the selected current impact analysis techniques (CIP-IPF technique)?" the Mean results from both approaches at the Group Statistics box is reviewed. The results show the SDP-CIA value is 0.9060 and the CIP-IPF approach value is 0.7927. This shows that the SDP-CIA value is higher than the CIP-IPF approach. Thus, the values reject the null hypothesis (H_0: The SDP-CIA does not improve on the CIP-IPF approach results) and accept the alternate hypothesis (Ha: The SDP-CIA approach gives higher

accuracy of impact analysis results than the CIP-IPF approach).

TABLE III
INDEPENDENT T-TEST RESULTS BETWEEN THE PATH IMPACT AND THE SDP-CIA

The Technique	Means Results
Path Impact	0.7773
SDP-CIA	0.9060

B. Stage 2 Analysis: The Path Impact Technique vs. The SDP-CIAF

Table III shows the Independent T-Test results. Similarly to the Independent T-Test between the CIP-IPF approach and the SDP-CIA approach, the null hypothesis for the Independent T-Test was that the Kappa mean value from both approaches are equal, H_0: $\mu 1=\mu 2$. The null hypothesis is accepted if the Significance (2-tailed) value is greater than 0.05. The alternative hypothesis was used to reject the null hypothesis if the Kappa means values from both approaches are not equal, H_0: $\mu 1\neq\mu 2$. The alternative hypothesis is accepted if the Sig. (2-tailed) value or the rho value is less than 0.05. Thus, the SDP-CIA approach gives higher accuracy of impact analysis results than the selected current impact analysis approach (in particular to the Path Impact technique).

To answer the question of "Does the SDP-CIA give better accuracy of impact analysis results than the selected current impact analysis techniques? (Path Impact technique)?", the Mean values from both approaches at the Group Statistics box is reviewed. The results show the SDP-CIA approach value is 0.9060 and the Path Impact approach value is 0.7773. This shows that the CIP-IPF approach value is higher than the Path Impact approach. Thus, the values reject the null hypothesis (H_0: The SDP-CIA does not give higher accuracy of impact analysis results than the Path Impact approach) and accept the alternate hypothesis (H_a: The SDP-CIA approach gives higher accuracy of impact analysis results than the Path Impact approach).

VI. CONCLUSION

This paper contributes a new approach that can be used for performing impact analysis during software development through partially developed artifacts consideration in its analysis.. This approach combines current static and dynamic analysis techniques, and supplements actual class interactions derived from source code with inferred class interactions derived from the requirements.

ACKNOWLEDGMENT

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] S. A. Bohner and R. Arnold, *Software Change Impact Analysis*, Wiley-IEEE Computer Society Press, July, 1996.
[2] J. Hassine, J. Rilling, J. Hewitt, and R. Dassouli, "Change Impact Analysis for Requirement Evolution using Use Case Maps," in *Proc. of the 8th International Workshop on Principles of Software Evolution*, 5-6 Sept. 2005, pp. 81 – 90.
[3] M. Shiri, J. Hassine, J. Rilling, "Feature Interaction Analysis A Maintenance Perspective," in *Proc. of the 22nd IEEE/ACM International Conference on Automated Software Engineering*, November 2007, pp. 437-440.
[4] Y. Li, J. Li, Y. Yang, and M. S. Li, "Requirement-centric traceability for change impact analysis: A case study," in *Proc. International Conference on Software Process (ICSP 2008)--Making Globally Distributed Software Development a Success Story*, , Leipzig, Germany, May 10-11, 2008, pp. 100-111.
[5] J. Law and G. Rothermal, "Whole Program Path-Based Dynamic Impact Analysis," in *Proc. of the 25th International Conference on Software Engineering (ICSE 2003)*, May 2003, pp. 308-318.
[6] T. Apiwattanapong, A. Orso, and M. J. Harrold, "Efficient and precise dynamic impact analysis using execute-after sequences," in *Proc. of the 27th Int. Conf. on Software Engineering*, May 2005, pp. 432-441.
[7] A. Orso, T. Apiwattanapong, and M. J. Harrold, "Leveraging field data for impact analysis and regression testing," in *Proc. of the ACM SIGSOFT Symposium on Foundations of Software Engineering*, September 2003, pp. 128-137.
[8] B. Breech, M. Tegtmeyer, and L. Pollock, "Integrating influence mechanisms into impact analysis for increased precision," in *Proc. of the 22nd International Conference on Software Maintenance*, September 2006, pp. 55-65.
[9] K. H. Bennet, V. T. Rajlich, "Software maintenance and evolution: A roadmap," in *Proc. of the Int. Conf. on the Future of Software Engineering*, June 2000, pp. 75-87.
[10] B. Nuseibeh, and S. Esterbrook, "Requirement engineering: A roadmap," in *Proc. of the Conference on the Future of Software Engineering (ICSE)*, June 2000, pp. 35-46.
[11] R. S. Arnold and S. A. Bohner, "Impact analysis - Towards a framework for comparison," in *Proc. of the Int. Conf. on Software Maintenance*, September 1993, pp. 292-301.
[12] S. Horwitz, T. Reps, and D. Binkley, "Interprocedural slicing using dependence graphs," *ACM Transactions on Programming Languages and Systems*, vol. 12, no. 1, July, 1998, pp. 26-60.
[13] J. S. O'Neil, and D. L. Carver, "Analyzing the impact of changing requirements," in *Proc. of the IEEE International Conference on Software Maintenance*, November 2001, pp. 190-195.
[14] N. Kama, T. French, and M. Reynolds, "Impact analysis using class interactions prediction approach," in *Proc. of the 9th International Conference on New Software Methodologies, Tools and Techniques*, October 2.
[15] O. Gotel, Contribution Structures for Requirements Traceability, PhD. Imperial College of Science, Technology and Medicine, Department of Computing, University of London, August 1995.

[16] A. L. Pfleeger, S. A. Bohner, "A framework for software maintenance metrics," in *Proc. of the Int. Conference on Software Maintenance*, November 1990, pp. 320-327.
[17] G. Spanoudakis, "Plausible and adaptive requirements traceability structures," in *Proc. of the 14th International Conference on Software Engineering and Knowledge Engineering*, July 2002, pp. 135-142.
[18] G. Spanoudakis, A. Zisman, E. P. Minana, and P. Krause, "Rule-based generation of requirements traceability Rrelations," *Journal of Systems and Software*, vol. 72, no. 2, July 2004, pp. 105–127.
[19] R. C. Sharble and S. S. Cohen, "The object-oriented brewery: A comparison of two object-oriented development methods," *ACM Software Engineering Notes*, vol. 18, no. 2, April 1993, pp. 60-73.
[20] A, Bahrami, Object Oriented Systems Development, McGraw-Hill
[21] Y. Liang, "From use cases to classes: A way of building object model with UML," *Journal of Information and Software Technology*, vol. 45, no. 2, February 2003, pp. 83-93.
[22] L. Huang and Y. T. Seong, "Dynamic impact analysis using execution profile tracing," in *Proc. of the 4th International Conference on Software Engineering Research, Management and Applications*, August 2000, pp. 237-244.
[23] L. Huang, and Y. T. Seong, "Precise dynamic impact analysis with dependency analysis for object-oriented programs," in *Proc. of the 5th ACIS International Conference on Software Engineering Research, Management & Applications*, August 2007, pp. 374-384.
[24] R. C. Metzger, Debugging by Thinking: A Multidisciplinary Approach, Elsevier Digital Press.
[25] J. Fung, C. Lau, E. Mckay, V. Birsan, C. Yu, J. Winchester, G. Mendel, and F. Flood, An Introduction to IBM Rational Application Developer: A Guided Tour (IBM Illustrated Guide Series), Mc Press.
[26] E. Dustin, Effective Software Testing: 50 Specific Ways to Improve Your Testing, Addison-Wesley.
[27] N. Kama, T. French, and M. Reynolds, "Considering patterns in class interactions prediction," In *Advances in Software Engineering*, Springer Berlin Heidelberg, vol. 117, pp. 11-22.
[28] N. Kam and F. Azli, "A change impact analysis for the software development phase," in *Proc. of the 19th Asia-Pacific Software Engineering Conference (APSEC 2012)*, December 2012, pp. 583-592.
[29] A. Finkelstein and J. Kramer, "Software engineering: A roadmap," in *Proc of the Conference on the Future of Software Engineering*, October 2000, pp. 3-22.
[30] N. Kama, T. French, and M. Reynolds, "Predicting class interactions from requirement interactions: Evaluating a new filtration approach," in *Proc. of the IASTED International Conference on Software Engineering*, February 2010, pp. 109-116.
[31] J. Cohen, "A coefficient of agreement for nominal scales," *Journal of Educational and Psychological Measurement*, vol. 20, no. 1, pp. 37-46, April 1960.

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, second degree in Real Time Software Engineering at the same university in 2003. He is currently pursuing his PhD at UTM and his research interest includes Software Engineering, Change Management and Effort Estimation.