Code Refactoring and its Impact on Internal and External Software Quality: An Experimental Study

Mohammed Alawairdhi

Saudi Electronic University, College of Computing and Informatics, Riyadh, Saudi Arabia

Summary
Over a significant period of time, code refactoring has evoved as an approach to improve effectiveness of legacy systems. Using refactoring, we can improve the internal structure of application without having to necessarily modify external behavior. This helps in improving quality of systems in terms of its reusability, scalability, understandability and flexibility etc. In this paper, the code of three components of an existing application under operation Saudi Electronic University has been refactored using seven refactoring techniques. The refactored components have been evaluated for selected internal and external quality measures. The results show a profound improvement in the internal quality of refactored components whereas the improvement in external quality measures has been found to be modest. An analysis of results has been made at the conclusion of the study.

Key words:
Refactoring; Software Quality; Efficiency; Reusability; Software Systems, Legacy Systems; Maintainability

1. Introduction

The purpose of refactoring is to modify the internal structure of a software component in such a way that it is more understandable and easiliy modifiable without making any change in its external behavior [1], This is an effective method of bringing change in a legacy system without compromising the functionality of system from external aspect. The number of bugs can be reduced significantly using refactoring. This also helps in improvement in internal structure, improved readability, maintainability and reduced complexity.
Software systems by their core nature need to evolve. Research has shown that over period of time, maintenance is a major cost, sometimes consuming up to 90% of overall software spending [2]. It has been observed that one effective method of preserving software quality without increasing maintainability cost is by using code refactoring techniques. In recent past, several studies have been conducted to examine the impact of refactoring on software quality [3.4]. However as the critique of these works in [5. 6] highlights, the studies lack empirical evidence of the impact of refactoring on software quality.
Refactoring can be performed manually or with the help of some software. Manual refactoring in today's world though still in practice can be error prone and time consuming. Automatic refactoring while efficient can also guarantee preserving system behavior which is a pre-requite for code refactoring. Automated refactoring comprises of two steps [7]. First step involves analysis to determine if refactoring preconditions hold in components being planned to be refactored. If first steps holds true, in second step, actual transformation of code takes place. Some concepts related to refactoring are as follows;
Bad Smells: The term bad smell refers to design flaws in the code. These are considered problems in the code which can be removed using refactoring. A few examples of Bad smell can be Large Class, Long Method, Duplicate Code etc.
Refactoring Techniques: These techniques used to remove bad smells and make the code clean using refactoring. Literature presents dozens of techniques for code refactoring. Some more prominent ones include Extract Method, Inline Method, Move Method, Inline Class, Rename Method etc.
In this paper, effort has been made to quantitatively evaluate performance of leading refactoring techniques on the quality of code. This study would help us evaluate on experimental level if cost and time invested refactoring would have been used for meaningful advantage. The study has been carried out on code components of an existing academic system at Saudi Electronic University.
Remaining paper is arranged as follows. After this brief introduction, a literature review is presented in section 2. Section 3 is dedicated to describe the Research methodology. Section 4 is dedicated to presenting results and discussion of those results. Paper concludes with future work in section 5.

2. Literature Review

A growing number of studies in recent years have focused on understanding the impact of refactoring on software quality. Unfortunately very few of such studies have presented any quantitative empirical evidence of this effect. Most studies Most of the studies have relied on qualitative and literature oriented discussion. Martin Fowler is considers one of the earliest proponents of code refactoring, In his paper [1], he equate code refactoring in object oriented development as equivalent to restructuring in more classical models of development. In the same

Manuscript received June 5, 2019
Manuscript revised June 20, 2019

paper, Fowler discussed the concept of bad smells. The two steps of refactoring were proposed by Roberts [7] where first is analysis and second is execution.

When coming to quality assessment of code refactoring. Bois and Mens proposed a metrics based technique to evaluate refactoring impact of quality of software [8]. The results of their work were inconclusive as those were without any experimental evaluation. Stroggylos and Spinellis [6] studied and analyzed impact of change due to refactoring on the source code of four open source software. They concluded that refactoring had no measurable impact on quality of the system. Bois et al. [9] developed guidelines for coupling in order to improve cohesion and coupling. They applied these guidelines on an open source software.

Kannangara and Wijayanayake [10] evaluated ten refactoring techniques to understand their impact on software code with respect to five quality measures. Their study showed that code refactoring had visibly positive impact on maintainability. In another study, Kannangara and Wijayanayake [11] again applied ten refactoring techniques and evaluated their impact on four different quality attributes. However their study found no noticeable improvement. Moser et al. [12] proposed a methodology to study thhe impact of refactoring on software reusability. Their study found that code refactoring has a visible positive impact of software reusability, The study carried out by Alshayeb [13] quantitatively assessed the impact of code refactoring on software quality. However this study as well failed to establish any positive correlation between code refactoring and software quality. Shatnawi and Li [14] applied several dozen refactoring techniques on two open source software systems to analyze their impact on software quality. Their study empirically found that most refactoring techniques have positive impact on software quality. However, they also noted that some refactoring techniques don't have any visible positive impact of external quality of the software.

As of the recent most exciting developments in the domain of refactoring, Chapparo et al [20] propsoed a method called RIPE (Refactoring Impact Prediction) to estimate the impact of refactoring techniques on quality of source code. Dallal et al. [16] presented an empirical literature review of several dozen previous studies summarizing the impact of code refactoring on software quality. Their findings indicated that application of different refactoring techniques produced opposite quality results for refactored components. Dallal [17] also developed a framework to evaluate the impact of various primary studies carried out on evaluating impact of refactoring on quality of software application. Jason in his work [18] worked on application which was being modified and refactored simultaneously. His results indicated positive effects of refactoring on application quality measures. Ouni Ali et al. [19] proposed a multi-criterion refactoring approach where refactoring was carried out using different techniques simultaneously in an automated manner. Their results showed positive impact of refactoring on software cyclomatic complexity.

To summarize all of the work in a nutshell, we come up with following understandings about the impact of code refactoring on software quality from literature perspective;

1. There is no definite conclusion yet about whether code refactoring has a positive impact on software quality and to what extent.
2. Most of the studies have either focused on internal quality aspect of external quality aspect. No study has covered both aspects of quality in a comprehensive manner.
3. Number of components evaluated were not representative in magnitude or quantity of the software systems from which those were extracted.

Keeping these factors in mind, we have used significant amount of components from application for case study. We have experimented with both internal and external quality aspects with significant number of refactoring techniques

3. Research Methodology

Selected Refactoring Techniques

As mentioned earlier, the purpose of this study was to examine the impact of code refactoring on both internal and external quality factors in a quantitative manner. The second major objective of this study was to measure the impact with as many software components as possible. In this section, the research method adopted would be explained in sufficient detail.

Fowler in their work [1] identified 72 refactoring techniques. Out of these, following techniques have been used in this study;

1. Duplicate Observed Data
2. Replace Type Code with Subclass
3. Replace Conditional with Polymorphism
4. Extract Subclass
5. Extract Interface
6. Form Template Method
7. Introduce Null Objects

Code refactoring is considered effective when it is applied to object oriented environment. Keeping this consideration in mind, we chose an application which was designed in PhP. The application used as case study is the student management system currently in use in Saudi Electronic University. Three components of this system were extracted for experimentation. These components were (1) Course Withdrawal (2) Semester Registration and (3) Transcript Generation. Source code comprised of approximately 2850 LOC and bad smells were pre-identified.

External and Internal Quality Metrics

External quality measures used to evaluate the impact of code refactoring on software systems were selected from ISO Quality model [15], Four quality measures were identified and studied which include;

Maintainability: The degree of effort needed to make specified modifications.

Efficiency: The quality attribute representing relationship between desired level of performance and resources utilized to maintain that performance.

Reusability: This quality attribute refers to the capacity of system or components of the system to be reused in other applications of systems

Performance: This quality attribute refers to the responsiveness of the system to perform given task in allocated time interval.

To measure the impact of code refactoring on internal quality of software, following six code measures were used;

Cyclomatic Complexity: Number of linearly independent paths through the application code. Less cyclomatic complexity means better designed source code.

Weighted Methods per Class (WMC): This metric is the sum of complexities of all class methods in given software component.

Lack of Cohesion of Methods (LCOM): This metric measures whether any given class in source code represents a single abstraction or multiple abstractions,

Depth of Inheritance Tree: This metric measures the maximum length from root to the node of tree in give source code or application or its component.

Class Coupling: This metric measures the coupling to unique classes through parameters, local variables, return types, method calls etc.

Line of Code: This metric counts the lines of code excluding empty lines and comments (containing actual source code).

4. Experimental Design

Experimentation for Internal Quality Measures:

Experimentation on each of three components was performed in following steps;

1. For each component
2. Run the legacy Program
3. Measure the complexity and other internal quality measures before making changes
4. Identify code for refactoring
5. Apply appropriate refactoring technique
6. Test refactored code to check if any error was introduced
7. In case of error, return to legacy program and correct the error. This would be followed by applying refactoring technique again and following step 5.
8. Repeat process for all refactoring techniques.
9. Measure internal quality metrics on refactored code.
10. Repeat from step 2-9 for each component.

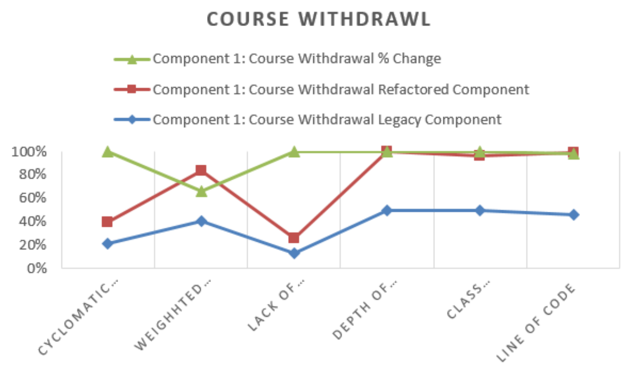

Fig. 1 Impact of Refactoring on Internal Quality Measures for Course Withdrawal Module

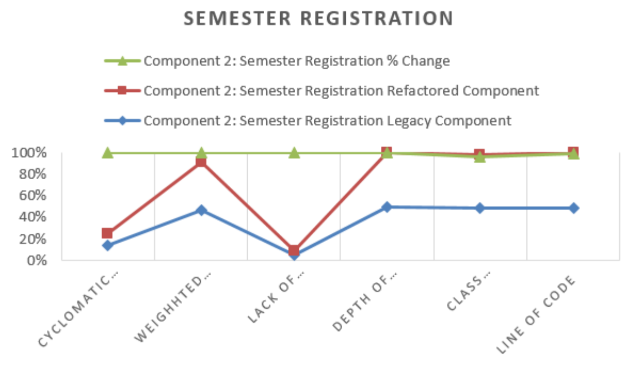

Fig. 2 Impact of Refactoring on Internal Quality Measures for Semester Registration Module

Experimentation for External Quality Measures

(a) Maintainability: A threshold was defined for effort required to make one change. Two changes in each component (legacy as well as refactored)were performed by similar software engineer and effort was required and compared against threshold.

(b) Efficiency: Time required to learn functionality by a novice user was calculated for legacy code as well as refactored code for each software component.

(c) Reusability: Percentage code candidate for reusability in legacy code as well as refactored code of each application component was recorded and compared.

(d) Performance: Similar used was asked to work with application component with legacy code as well as refactored code. Number of errors reported by the user for both was compared. The process was repeated for each of three components.

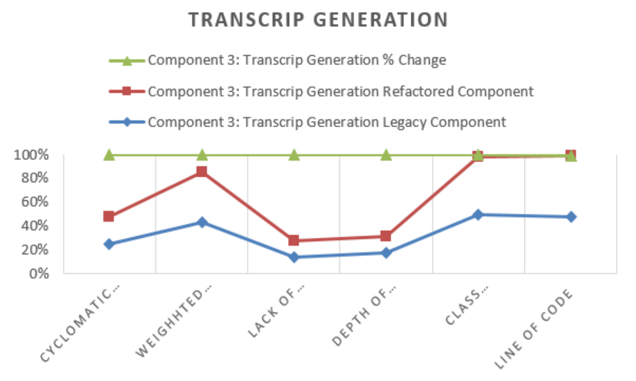

Fig. 3 Impact of Refactoring on Internal Quality Measures for Transcript Generation Module

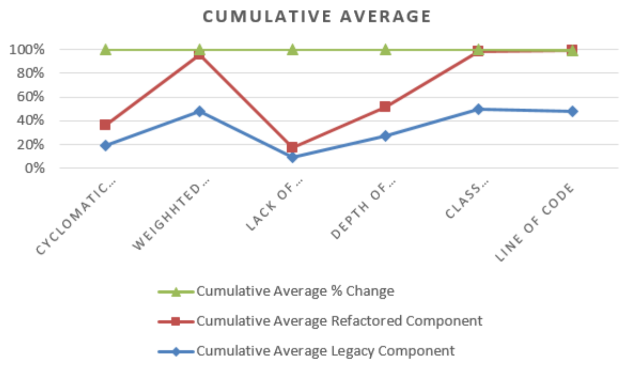

Fig. 4 Cumulative Average of impact of Refactoring on Internal Quality Measures for all Modules

5. Experimental Results and Analysis

In this section, results and findings of experiments performed according to experimental design are presented.

Results and Analysis for Internal Quality Measures

The results of experimentation performed with internal quality measures is shown in figure 1,2,3 and 4. each figure represents results for one module while figure 4 shows cumulative change for all internal quality measures. The experiments exhibit interesting information. It is evident from the table that code refactoring had the most profound impact on reducing the cyclomatic complexity. All the components which were refactored showed significant reduction in complexity with component 2 showing a complexity reduction of up to 20%.

One area of concern however was line of code which in all cases showed negative trend. It means that after refactoring, the Line of code for each component increased. This increase was most profound in the case of component one where an increase of 15% was registered. Evidence also showed that WMC for component 1 was also most adversely effected. It needs to be seen in future work if there exists a correlation between negative WMC at the end of code refactoring as increase in Line of code.

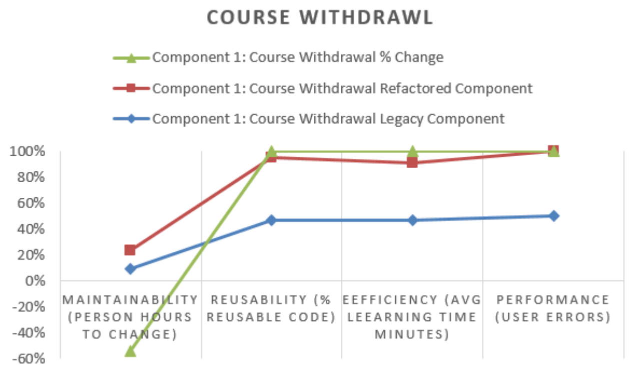

Fig. 5 Impact of Refactoring on External Quality Measures for Course Withdrawal Module

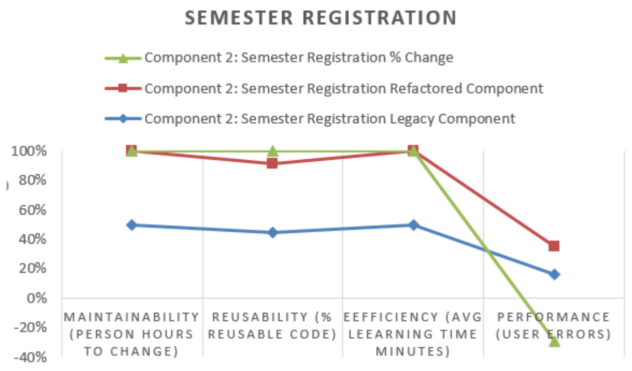

Fig. 6 Impact of Refactoring on External Quality Measures for Semester Registration Module

All other internal quality factors showed positive trend mostly with only two exceptions. Results showed that Weighted Methods Per Class actually increased for component 1 after code refactoring. Also Class coupling registered a modest increase after refactoring for component 2.

Another interesting discovery of experimentation was that code refactoring had minimal effect on Depth inheritance Tree. The depth inheritance Tree showed no change for component 1 and component 2. However for component 3, code refactoring was able to reduce depth by one level. When we look at overall accumulative effect of refactoring for various internal quality attributes, it is evident that code refactoring has most profound positive impact on cyclomatic complexity. It also has a positive impact on all other chosen quality measures except Line of Code which show a significant increase after code refactoring.

Results and Analysis for Internal Quality Measures

Results of applying code refactoring on external quality measures are shown in figure 5,6,7 and 8. These results showed an interesting picture. As is evident from the results, code refactoring didn't have any major impact on external quality attributes. Except for maintainability which was found to be requiring more effort for refactored code, all other quality attributes showed very minimal change. This change can very well be classified as within the margin of error. One other interesting pattern observed was that performance was reported to be slightly lower for the component which had high line of code for both legacy as well as refactored component. This may indicate that with increase in size of code, probability of making errors in the system increases during refactoring. Results found no major change in reusable components after refactoring and efficiency was also found to be approximately similar for both legacy as well as refactored components.

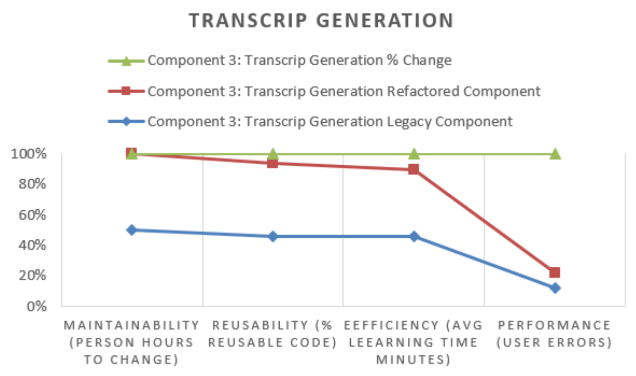

Fig. 7 Impact of Refactoring on Internal Quality Measures for Transcript Generation Module

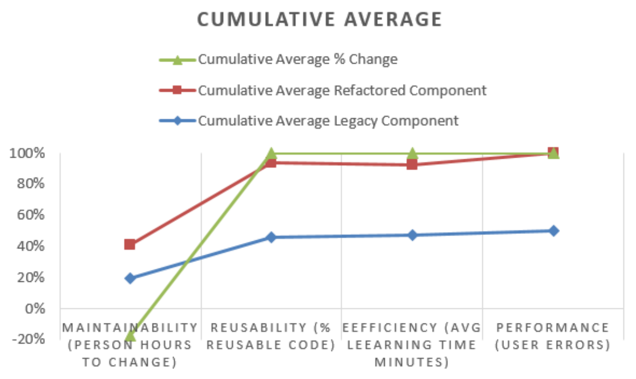

Fig. 8 Cumulative Average of impact of Refactoring on External Quality Measures for all Modules

One interpretation that can be drawn from these results is that code refactoring has a profound impact on internal quality measures as it significantly improves the internal structure of the application. However, for the external behavior, this impact is minimal because the focus is not on modifying the functionality of the system. Rather it is to restructure the legacy code to make it easier to change in future and maintain it. At the same time, the study can be cited as inconclusive since it covers only four external quality measures. In future, any new study can accommodate more external quality measures and reach to a more definitive conclusion.

6. Conclusion and Future Work

Code refactoring is an important approach for improving the quality of code in legacy systems. In recent years a lot of work has been done to catalogue various refactoring techniques. This has opened a new area of research where the quality of application or application components after refactoring is being examined. After going through literature, one comes across two limitations of current evaluation studies. One limitation is that most of studies rely on qualitative data rather than experimentation with refactoring techniques. Second limitation that studies either focus on Internal quality measures of external quality measures. No comprehensive study has appeared which shows satisfactory quantitative analysis for both external and internal quality measures and their possible relationships.

This paper presents an empirical study of impact of code refactoring on selected internal and external quality measures. Three component of legacy management system at Saudi Electronic University have been chosen and seven refactoring techniques have been applied on each of the component. After refactoring, each component has been quantitatively evaluated for six internal and 4 external quality measures. Results show that code refactoring has significant impact on internal quality aspects of application. However, the impact of code refactoring on external quality measures is not so profound,

In future, there is a need to expand the scope of current work by increasing the dataset for experimentation as well as increasing the number of refactoring techniques being applied for refactored components. Another important area of interest can be to include more external quality measures from ISO standards and deeply observe the behavior of application after refactoring. One more interesting area can be to apply both architectural as well as code refactoring on legacy systems and observe their impact on external as well as internal quality of the new refactored system.

Acknowledgments

The author will like to acknowledge and express his gratitude to Saudi Electronic University and Ministry of Education, Saudi Arabia for providing conducive environment in order to perform studies and research in various disciplines which has always been instrumental in carrying out my work.

References

[1] Martin Fowler, Kent, John Brant, William Opdyke, Don Roberts, D. B. “Refactoring: Improving the Design of Existing Code”, Addison-Wesley, New York, (1999).

[2] Koskinen J. (2010). Software Maintenance Costs. Jyväskylä: University of Jyväskylä.

[3] Kataoka Y, Imai T, Andou H, Fukaya T. A (2002). Quantitative evaluation of maintainability enhancement by refactoring. In Proceedings of the IEEE International Conference on Software Maintenance, Montreal, Quebec, Canada.

[4] Wilking D, Khan U, Kowalewski S. (2007). An empirical evaluation of refactoring, e-Informatica Software Engineering Journal, 1, 27-42.

[5] Mens T, Demeyer S, Bois B D, Stenten H, Gorp P V. (2003). Refactoring: Current Research and Future Trends. Electronic Notes in Theoretical Computer Science, 80(3).

[6] Stroggylos K, Spinellis D. (2007) Refactoring – does it improve software quality?, In Proceedings of 5th International Workshop on Software Quality (WoSQ’07:ICSE Workshops), 10–16.

[7] Roberts, D. B “Practical Analysis for Refactoring”, PhD thesis, Department of Computer Science , University of Illinois at Urbana-Champaign, (1999).

[8] Bois B.D, Mens T. (2003). Describing the impact of refactoring on internal program quality, In Proceedings of the 8. International Workshop on Evolution of Large-scale Industrial Software Applications, Amsterdam, The Netherlands, 37-48.

[9] Bois B D, Demeyer S, Verelst J. (2004). Refactoring – improving coupling and cohesion of existing code. In Proceeding of the 11th Working Conference on Reverse Engineering (WCRE’04), 144–151.

[10] S. H. Kannangara and W.M.J.I. Wijayanayake, “Measuring the Impact of Refactoring on Code Quality Improvement Using Internal Measures”, In Proc. of the International Conference on Business & Information, Sri Lanka, December 2013.

[11] S. H. Kannangara and W.M.J.I. Wijayanayake, “Impact of Refactoring on External Code Quality Improvement: An Empirical Evaluation”, In Proc. of International Conference on Advances in ICT for Emerging Regions, Sri Lanka, December 2013.

[12] Moser R, Abrahamsson P, Pedrycz W, Sillitti A, Succi G. (2007). A case study on the impact of refactoring on quality and productivity in an agile team. In Proceeding of the Central and East-European Conference on Software Engineering Techniques, Poznan, Poland.

[13] Alshayeb M. (2009). Empirical investigation of refactoring effect on software quality. Information and Software Technology Journal, 51(9), 1319–1326.

[14] Shatnawi R, Li W. (2011). An Empirical Assessment of Refactoring Impact on Software Quality Using a Hierarchical Quality Model, International Journal of Software Engineering and Its Applications, 5(4).

[15] ISO/IEC 9126-1 Standard. (2000). http://www.cse.unsw.edu.au/~cs3710/PMmaterials/Resources/9126-1%20Standard.pdf. Accessed 10 July 2014.

[16] Al Dallal, Jehad, and Anas Abdin. "Empirical Evaluation of the Impact of Object-Oriented Code Refactoring on Quality Attributes: A Systematic Literature Review." IEEE Transactions on Software Engineering (2017).

[17] Al Dallal, Jehad. "Evaluating quality of primary studies on determining object-oriented code refactoring candidates." Proceedings of the The International Conference on Engineering & MIS 2015. ACM, 2015.

[18] Jonsson, Alan. "The Impact of Refactoring LegacySystems on Code Quality Metrics." (2017).

[19] Ouni, Ali, et al. "Multi-criteria code refactoring using search-based software engineering: An industrial case study." ACM Transactions on Software Engineering and Methodology (TOSEM) 25.3 (2016): 23.

[20] Chaparro, Oscar, et al. "On the impact of refactoring operations on code quality metrics." Software Maintenance and Evolution (ICSME), 2014 IEEE International Conference on. IEEE, 2014.

Mohammed Alawairdhi received his Bachelor degree in information Systems from King Saud University, Saudi Arabia. Then he received his Master degree in Computer Science from California State University, Chicago, USA. He earned his Ph.D. degree in Computer Science from De Montfort University, UK in 2009. He works as an Assistant Professor in the college of Computer and Information Sciences, Al - Imam Muhammad ibn Saud Islamic University. He was also vice dean of Graduate Studies and Research, College of Computer and Information Sciences until June 2012. He is currently working as vice president of research and higher studies at the Saudi Electronic University. His research interests include software engineering, ubiquitous computing, business process re-engineering, computational intelligence and human computer interaction (culture and cyberspace).