Impact of Design Patterns on Software Maintainability

Article *in* International Journal of Intelligent Systems and Applications · September 2014

DOI: 10.5815/ijisa.2014.10.06

2 authors:

Fatimah Alghamdi
Durham University
15 PUBLICATIONS **12** CITATIONS
SEE PROFILE

M. Rizwan Jameel Qureshi
King Abdulaziz University
117 PUBLICATIONS **917** CITATIONS
SEE PROFILE

I.J. Intelligent Systems and Applications, **2014, 10, 41-46**
Published Online September 2014 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijisa.2014.10.06

Impact of Design Patterns on Software Maintainability

Fatimah Mohammed Alghamdi, M. Rizwan Jameel Qureshi
Faculty of Computing and Information Technology, King Abdulaziz University, Jeddah, Saudi Arabia
Emails: f_t_m.g@hotmail.com, anriz@hotmal.com

***Abstract*—** This paper mainly studies the effect of design patterns on the Software maintainability. Design patterns describe solutions for common design problems and they were introduced to improve software quality and accelerate software development. However, there are some difficulties to choose an optimal pattern adapted to a certain application and problem. So until now the results on the effect of design patterns on software quality are controversial. In this context, we propose a tool for design pattern guided that retrieves the appropriate pattern with respect to software maintainability from a repository of patterns. It measures the maintainability of design pattern by some metrics and candidate the more maintainable pattern to the designer or developer. It provides a support for decision making during system design and refactoring. As the results, the decision of applying a certain design pattern is usually a trade-off since the effect of design pattern on software maintainability is influenced by some factors such as the pattern size and the prior expertise of the developer.

***Index Terms*—** Design Patterns, Software Maintainability, Metrics, Pattern Size, Tool

I. Introduction

A design pattern is a general reusable solution to a commonly occurring problem in software design. It can be defined as a description or template for how to solve a problem that can be used in many different situations [1]. There are three main types of design patterns that are architectural patterns, Gang of Four *(*GoF*)* design patterns and idiom patterns. In this paper we focus in the GoF design patterns that are cataloged in the widely referenced book by the "Gang of Four" [2]. The authors classified 23 patterns according to the purpose and according to the scope. The purpose reflects what a pattern does; patterns can have creational, structural, or behavioral purpose. The scope classification specifies whether the pattern applies primarily to classes or to objects. In [2], the authors suggest that using design patterns provide easier maintainability and reusability, more understandable implementation and more flexible design. In recent years, many researchers have attempted to evaluate the effect of *GoF* design patterns on software maintainability, they conducted several empirical methods such case studies, surveys and experiments, but safe conclusions cannot be drawn since the results lead to different directions. A design pattern needs to be investigated before it is used and the designers are expected to have a good understanding and experience with design patterns. In this situation, some problem still face the experienced designer which is time consuming to understanding, identification and investigation of the design pattern appropriate to his applications [3].

In this paper we have attempt to evaluate the effect of GoF design patterns on software maintainability to draw safe conclusion about this issue. We have proposed a tool to investigate which of design provide easier maintainability under considering the most common factor which is the system size. This tool helps the experienced and even the inexperienced designer for choosing the more maintainable pattern because it is supplied by a repository of patterns.

In the next section, we review the most recently related works. In Section III, the problem statement is stated, and proposed solution is summarized at Section IV. Then a validation of this solution is presented in Section V. Conclusion is given in Section VI.

II. Related Work

Design patterns have been subjected to limited empirical evaluation, and that much of this has also only been studying patterns indirectly [1]. Until now, researchers attempted to investigate the outcome of design patterns with respect to software quality through empirical methods, i.e. case studies, surveys and experiments, but safe conclusions cannot be drawn since the results are controversial [4].

The original study to evaluate the impact of design patterns on software maintenance was applies by Prechelt et al. [5]. They conducted an experiment called PatMain by comparing the maintainability of two implementations of an application, one using a design pattern and the other using a simpler alternative. They used four different subject systems in same programming language. They addressed five patterns: Decorator, Composite, Abstract Factory, Observer and Visitor. The researchers measured the time and correctness of the given maintenance tasks for professional participants. They found that it was useful to use a design pattern but in case where simple solution is preferred, it is good to follow the software engineer common sense about whether to use a pattern or not, and in case of uncertainty, it is better to use a pattern as a default approach. A thorough understanding of specific design patterns is often helpful for program maintenance.

PatMain experiment [5] replicated by Prechelt and Liesenberg [6] but in much reduced form. They used two

systems out of the four used in the original experiment and in different programming languages. The participants were 13 students. Their results confirmed the result of the original experiment but due to the small size of the experiment they found only one statistically significant result: the non-pattern based version of one of systems was more maintainable and can be extended more quickly.

Juristo and Vegas [7] conducted another replication study for PatMain experiment [5]. They conducted their study on two software systems in two different languages. They addressed three different patterns: Abstract Factory, Composite and Decorator. The participants were 8 master students. The dependent variable was only the time (in minutes) to complete each maintenance task. Their results were inconsistent with the original study. They found that systems with design patterns were less maintainable.

Nanthaamornphong and Carver [8] also replicated the PatMain experiment [5]. In their experiments they used the same systems of the original experiment. They focused on four patterns: Observer, Visitor, Decorator and Composite. Eighteen students in a graduate-level Software Engineering course participated in the study. The results of this replication were different from those in the original study. They found that the design patterns did not improve either the maintainability or the understandability of these systems.

Krein et al. [9] performed also a replication for the same experiment done by Prechelt et al. [5]. In this experiment they used two systems in two different languages. They studied three different patterns: Decorator, Composite and Abstract Factory. They found that by performing some modifications on the two versions, the pattern version and the non-pattern version, the pattern based designs took longer time and have more faults than non-pattern designs except for one modification task.

Hegedus et al. [10] evaluated the impact of design patterns on maintainability directly by conducting an empirical analysis. They analyzed more than 300 revisions of the JHotDraw software system which relies heavily on some design patterns. They calculated the maintainability values with their probabilistic quality model and mined the design pattern instances parsing the comments in the source code. They calculated the maintainability values with their probabilistic quality model and mined the design pattern instances parsing the comments in the source code. They found that there is a strong relation between the rate of design patterns in the source code and the maintainability. Therefore using design patterns improve the code maintainability.

Zhang and Budgen [1] conducted a systematic literature review in the form of a mapping study to examine the extent and form of the empirical knowledge that is available for *GoF* design patterns. They augmented their analysis by including some "experience" reports that described application of patterns using less rigorous observational forms. They found some support for the usefulness of patterns in providing a framework for maintenance but they could not identify firm guidelines about the efficient use of particular patterns to improve the software quality because the available studies were inadequate.

Ali and Elish [11] performed a literature survey to understand the impact of the *GoF* design patterns on software quality attributes by comparing the existing empirical evidence in the literature. They investigated the impact of design patterns on four quality attributes: maintainability, evaluation, performance and fault-proneness. The results show that in general, the impact of design patterns on maintainability, evolution and change proneness is negative. For performance, the number of studies that addressed performance and the number of covered patterns make it difficult to draw a conclusion. Finally for fault- proneness, the results are different from one study to the other, thus it is difficult to make a decision regarding their impact.

Hsueh et al. [12] proposed an analytical assessment to evaluate the effectiveness of design patterns to help programmers to inspect the correctness of the application of these design patterns. They also proposed two different measurement ways for the application of design patterns: Occasion and effectiveness analysis to evaluate some well-known open source systems. They defined their context and their anticipated changes and then checked whether they held up to the expectations. Their conclusion provides that although design patterns can be misused, they are effective to some degree in either early stage or late stage of maintenance.

Nadia et al [3] presented approach assists the designers choosing their appropriate design patterns. Their approach was supported by an interactive tool and was guided by set of comparison criteria and recommendation rules. The tool allows the designer to draw a design fragment, present the problem then re-phrases the problem in order to obtain the intention of a certain pattern. Then, it explores the candidate solutions by filtering patterns that meet the intentions through the use of recommendation rules.

Ampatzoglou et al. [4] conducted study to propose a theoretical methodology by comparing three design patterns with two alternative solutions, with respect to several quality attributes, through the mathematical formulation and well known metrics. They investigated designs by studying the literature, open-source projects and by using design patterns. They have created decision support tool that aids the developer to choose the appropriate design pattern. The input of the tool is the pattern under consideration, the estimated system size and the goals of the design team with respect to quality attributes. The tool simulates all the steps of the proposed methodology. The results show that the decision of applying a design pattern is usually a trade-off because patterns are not universally good or bad, but it should be preferred for systems that are intended to be heavily reused and/or maintained. Furthermore, two additional factors have been highlighted: pattern size and developers' prior experience with pattern.

Table 1 gives a brief description for the related works regarding some limitations which are found in them.

Table 1. Summarization for the related works

Title of Paper	Limitations
Design Patterns in Software Maintenance: An Experiment Replication at Freie University at Berlin [6].	• The experiment is not described in enough detail, having missed important information, such as: o why particular software artifacts selected o why particular design patterns addressed o why a new programming language is added • Their results produce conflict to identify the real impact of design patter. • Not provide clear decision to select the efficient design pattern.
Design Patterns in Software Maintenance: An Experiment Replication at UPM - Experiences with the RESER'11 Joint Replication Project [7].	
Design Patterns in Software Maintenance: An Experiment Replication at University of Alabama [8].	
Design Patterns in Software Maintenance: An Experiment Replication at Brigham Young University [9].	
Myth or Reality? Analyzing the Effect of Design Patterns on Software Maintainability [10].	• It analyzed only one system with a relatively few number of patterns. • Its result should be handled with caution.
What Do We Know about the Effectiveness of Software Design Patterns? [1].	• The survey is need for more design-centric evidence. • The undertaken studies identified a small number of design patterns. • Not provide clear decision to select the efficient design pattern.
A Comparative Literature Survey of Design Patterns Impact on Software Quality [11].	• The undertaken studies have several variable factors that could produce differences in their results. • Not all the GoF design patterns were covered in the literature. • Its result should be handled with caution.
An Approach for Evaluating the Electiveness of Design Patterns in Software Evolution [12].	• Not provide clear way to select the appropriate design pattern.
A design pattern recommendation approach [3].	• Mixture between detection and select pattern.
A methodology to assess the impact of design patterns on software quality [4].	• The method cannot be applied to all design patterns.

III. Problem Statement

Which of the design patterns improve the software maintainability, and under what factors?

IV. Proposed Solution

Design patterns are not universally good or bad as the previous authors suggested in their empirical studies [5,9], but until now there is no study that identifies which of design patterns improve the software maintainability and which of them has weaken effect. The effect of design patterns on the software maintainability is governed by different factors such as pattern size, prior expertise of the developer with pattern and the most important quality attributes that must achieved by pattern [4], and before all of these is fitting the pattern to a certain design problem [3]. In [4] the authors have created a decision support tool that helps the developer to choose between three of *GoF* design patterns and equivalent alternative design solutions, it calculates metrics scores of each solution based on the system size, then it presents where a design solution is getting better than another with respect to several quality attributes. This paper have proposed a new version of this tool that aims to compare the maintainability of *GoF* design patterns with each other based on the maintainability predictors.

A. Design Patterns under Consideration

Design patterns can be maintained in three possible ways [13] which are adding a class as a concrete participant, modifying the existing interface participants or introducing a new client, and the first one is the most common type of maintenance according to that study [13]. So this way is selected to maintain the system and accordingly the axes of change were chose. The major axes of change in the design pattern [13] are: adding refined abstractions, adding concrete implementers, adding clients and adding methods and attributes to any class of pattern. I have chosen to extend/maintain the system in the first two axes, i.e. add new refined abstractions and add new concrete implementers. These axes are base for construct the equations of the metrics that used for comparing. At this point it is suitable to clarify that proposed tool provides for comparing design pattern with its alternative patterns that describe equivalent functionality and have specified axes of change. So according to the selected axes, the patterns under consideration are all *GoF* patterns that involve class hierarchies and client classes, shown in table 2. Theses pattern are gathered by inspecting the class diagram for each one as presented in the standard form according to *GoF* book [2]. Also these patterns are categorized such each one put with its alternative which share same functionality according to the *GoF* purpose classification [2].

B. Metrics as measurement of maintainability

There are ten object oriented metrics used as maintainability predictors [14,15] to investigate the effect of design pattern, all these metrics defined in table 3. Each metric has constructed equation based on the selected axes of change; hence the comparison is done by

calculating the equations and comparing the result values. The pattern with the higher count of lower metric values is considered more maintainable [16,17].

Table 2. Design pattern under consideration

Creational	Structural	Behavioral
Abstract Factory Builder Prototype	Bridge Composite Decorator Flyweight Proxy	Interpreter Chain of Responsibility Observer State Strategy Visitor

Table 3. Maintainability predictors

Metric	Description
DIT	Depth of the inheritance tree (=inheritance level number of the class, 0 for the root class). Range of value [0,+1)
NOC	Number of children (=number of direct sub-classes that the class has). Range of value [0,+1)
MPC	Message-passing couple (=number of send statements defined in the class). Range of value [0,+1)
RFC	Response for a class (=total number of local methods and the number of methods called by local methods in the class). Range of value [0,+1)
LCOM	Lack of cohesion of methods (=number of disjoint sets of local methods, i.e. number of sets of local methods that do not interact with each other, in the class). Range of value [0,+1)
DAC	Data abstraction coupling (=number of abstract data types defined in the class). Range of value [0,+1)
WMPC	Weighted method per class (=sum of McCabe's cyclomatic complexity of all local methods in the class). Range of value [0,+1)
NOM	Number of methods (=number of local methods in the class). Range of value [0,+1)
SIZE1	Lines of code (=number of semicolons in the class). Range of value [0,+1)
SIZE2	Number of properties (=total number of attributes and the number of local methods in the class). Range of value [0,+1)

C. Tooling

The proposed tool aims to help the designer/developer to choose the appropriate design pattern that produces more maintainable system. The input of the tool is the pattern under investigation and the estimated pattern size which is number of refined abstractions classes (n) and number of concrete implementers classes (m). The functional architecture of proposed tool is shown in figure 1, the user selects the pattern he wants to examine then selects the metrics he is interested in and finally defines the (n) and (m) for the pattern.

The tool retrieves all patterns that describe equivalent functionality from a repository of patterns, and then calculates the mathematic equations of selected metrics for each equivalent pattern. The tool displays the results in two phases: first phase indicates the average metric scores for each pattern in the given range of (n) and (m), and the second phase determines which pattern produces ''best'' results i.e. has the higher count of lower metric values then consider as more maintainable.

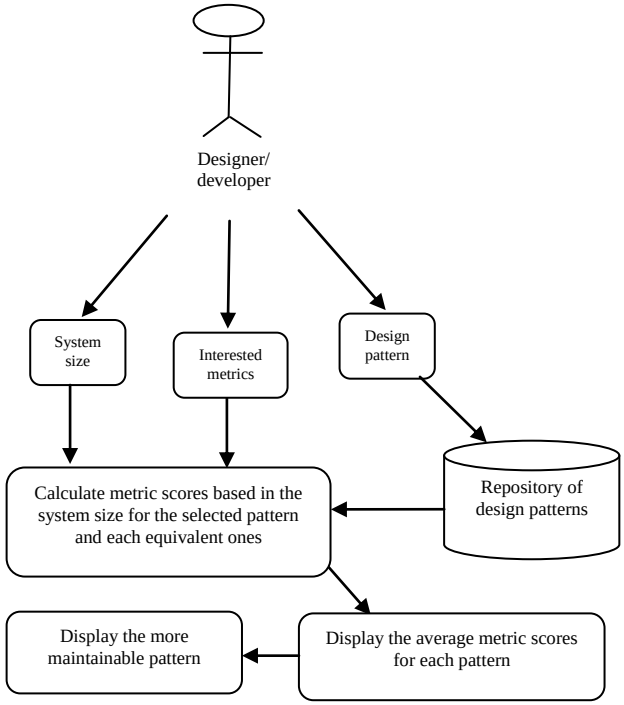

Fig. 1. Architecture of the proposed tool

V. Validation Of The Proposed Tool

Survey was conducted for the validation purpose. A questionnaire consisting of 17 close ended questions divided into 3 goals was used for data gathering on basis of a 5-point likert scale, which is given in table 4. Questions were arranged according to their relevancy to defined goals. We preferred to use an electronic survey because it is it's not take too much time and gives the respondent much of time to think and answer questions be credible, then we shared the link of that survey with some people who are specialized in the software engineering. Once the responders are collected they are statistically analyzed for cumulative evaluation to find support to our hypothesis or vice versa, as shown below.

Following are the three basic goals that divided questions in the electronic survey:

Goal 1: The necessity of the proposed tool

This goal provides the answers of the questions that will be explored the extent of the necessity of the proposed tool. The more maintainable design pattern makes the system easier in the maintenance, but there is some difficult to find the perfect pattern especially if the developer has not sufficient experience in the design patterns.

Table 4. Likert scale

1	Strongly Disagree
2	Disagree
3	Neutral- Neither Agreed Nor Disagree
4	Agreed
5	Strongly Agreed

Goal 2: The efficiency of the proposed tool

This goal has been presented into seven questions; the answers will help us to measure the efficiency of the proposed solution. The maintainability metrics are used to investigate the effect of design pattern on the software maintainability; this effect is influenced by the system size. The repository adds a positive effect for storing and retrieving the design patterns.

Goal 3: How to improve the proposed tool

This goal provides the answers of the questions that will be taken into account during the enhancement the proposed tool.

Table 5. Cumulative statistical analysis of all three goals

Q. No	Str. Disagree	Disagree	Neutral	Agree	Str. Agree
1	0	1	6	7	8
2		3	7	10	2
3			2	11	9
4		1	2	15	4
5		2	7	11	2
6		3	6	12	1
7			6	12	4
8	3	3	13	3	0
9		1	5	13	3
10		1	12	7	2
11		1	6	11	4
12			1	15	6
13		2	4	9	7
14			12	8	2
15		1	3	15	3
16	1	1	8	9	3
17		1	3	13	5
Total	4	21	103	181	65
Avg.	1.07%	5.61%	27.54%	48.40%	17.38%

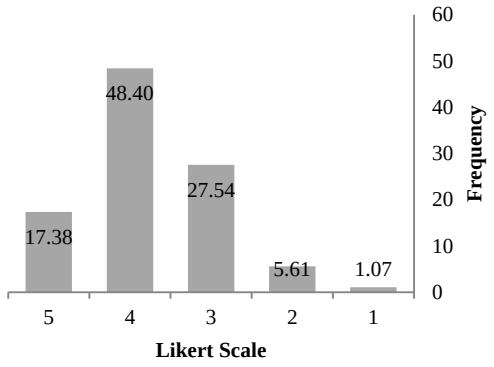

Fig. 2. Graphical representation of cumulative results for three goals

We evaluated all three defined goals as shown in table 5. The results are: 48.40% of the samples are agreed to the proposed solution and 17.38% are strongly agreed to it, whereas 5.61% are disagreeing and 1.07% are strongly disagreed. 27.54% of the sample are neither agreed nor disagree. These results are presented in figure 2.

VI. Conclusion

The authors proposed a solution to evaluate the effect of design patterns on software maintainability. This solution is simulated by a tool that measures the maintainability of each pattern by some relevant metrics with regard the system size. In fact, we realize the changes are frequent throughout the software development process and we expected the utilization design pattern will facilitate those changes during maintenance as it is reusable component. The results of survey proved that tool provides a good evaluation for the design pattern with respect to software maintainability. It helps the designer/developer to choose the appropriate design pattern that produces more maintainable system as it is observed by the respondents of questionnaire.

As the results, the decision of applying a certain design pattern is usually a trade-off since the effect of design pattern on software maintainability is influenced by some factors such as pattern size, prior expertise of the developer with pattern and the most important quality attributes.

Thus, future work includes a deeper research with the factors that control the effect of design patterns on software maintainability. Furthermore we plan to automate the methodology of tool to take the size information from an already implemented pattern, with respect to specific design quality attributes adding to available maintainability metrics in order to enhance the decisions of applying a certain design pattern.

References

[1] C. Zhang and D. Budgen, "What Do We Know about the Effectiveness of Software Design Patterns?," *IEEE Transactions on Software Engineering*, vol. 38, no. 5, Sep./Oct. 2012, pp. 1213- 1231.

[2] E. Gamma, R. Helms, R. Johnson, J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley Professional, Reading, MA, 1995.

[3] B. Nadia, A. Kouas and H. Ben-Abdallah, "A design pattern recommendation approach", *CORD Conference Proceedings*, pp. 590- 593, 2011.

[4] A. Ampatzoglou, G. Frantzeskou and I. Stamelos, "A methodology to assess the impact of design patterns on software quality," Information and Software Technology, Elsevier, vol. 54, no. 4, April 2012, pp. 331–346.

[5] L. Prechelt, B. Unger, W.F. Tichy, P. Brossler and L.G. Votta , "A controlled experiment in maintenance: comparing design patterns to simpler solutions," *IEEE Transactions on Software Engineering*, vol. 27, no. 12, Dec. 2001, pp. 1134-1144.

[6] L. Prechelt and M. Liesenberg, "Design Patterns in Software Maintenance: An Experiment Replication at Freie University at Berlin," *Second International Workshop on Replication in Empirical Software Engineering Research (RESER)*, Sept. 2011 pp.1-6, 21. DOI 10.1109/ RESER.2011.12

[7] N. Juristo, S. Vegas, "Design Patterns in Software Maintenance: An Experiment Replication at UPM - Experiences with the RESER'11 Joint Replication Project," *Second International Workshop on Replication in Empirical Software Engineering Research (RESER*), Sept. 2011, pp.7-14, 21. DOI 10.1109/RESER.2011.8

[8] A. Nanthaamornphong and J. C. Carver, "Design Patterns in Software Maintenance: An Experiment Replication at University of Alabama," *Second International Workshop on Replication in Empirical Software Engineering Research (RESER)*, Sept. 2011, pp.15-24, 21-21. DOI 10.1109/RESER.2011.11

[9] J.L. Krein, L. J. Pratt, A.B. Swenson, A.C. MacLean, C. D. Knutson, and D.L. Eggett , "Design Patterns in Software Maintenance: An Experiment Replication at Brigham Young University," *Second International Workshop on Replication in Empirical Software Engineering Research (RESER)*, Sept. 2011, pp.25-34, 21-21. DOI 10.1109/ RESER.2011.10

[10] P. Hegedűs, B. Dénes, F. Rudolf and G. Tibor, "Myth or Reality? Analyzing the Effect of Design Patterns on Software Maintainability."*Computer Applications for Software Engineering, Disaster Recovery, and Business Continuity* , 2012, pp. 138-145

[11] M. Ali and M.O. Elish, "A Comparative Literature Survey of Design Patterns Impact on Software Quality," *Information Science and Applications (ICISA), 2013 International Conference on* , vol., no., pp.1,7, 24-26 June 2013. doi: 10.1109/ICISA.2013.6579460.

[12] N.L. Hsueh, L.C. Wen, D.H. Ting, W. Chu, C.H. Chang, and C.S. Koong, "An Approach for Evaluating the Effectiveness of Design Patterns in Software Evolution," In: IEEE 35th Annual Computer Software and Applications Conference Workshops (COMPSACW), July 2011, pp. 315–320. DOI 10.1109/COMPSACW.2011.59.

[13] T.H. Ng, S.C. Cheung, W.K. Chan, Y.T. Yu, Do maintainers utilize deployed design patterns effectively?, IEEE Proceedings of the 29th International Conference on Software Engineering, IEEE Computer Society, 20–26 May 2007, Washington, USA, pp.168-177.

[14] A. Van Koten, A.R. Gray, An application of bayesian network for predicting object-oriented software maintainability, Information and Software Technology, Jan. 2006, pp. 59–67.

[15] U. Zdun, P. Alexiou, C. Hentrich, S. Dustdar, Architecting as decision making with patterns and primitives, Proceedings of the 3rd International Workshop on Sharing and Reusing Architectural Knowledge (ICSE'08), IEEE, Leipzig, Germany, 10–18 May 2008, pp. 11–18.

[16] B. Henderson-Sellers, L. Constantine, I. Graham, Coupling, cohesion: towards a valid metrics suite for object-oriented analysis and design, Object-Oriented Systems, Mar. 2002, pp. 143–158.

[17] M. Lorenz, J. Kidd, Object-Oriented Software Metrics, Prentice Hall, New Jersey, USA, 2004.

Authors' Profiles

Fatimah Alghamdi is a graduate student of Information Technology, Faculty of Computing & Information Technology, King Abdul-Aziz University, major in Information Technology.

Dr. M. Rizwan Jameel Qureshi received his Ph.D. degree from National College of Business Administration & Economics, Pakistan 2009. This author is best researcher awardees from Department of Information Technology, King

Abdulaziz University Saudi Arabia in 2013 and Department of Computer Science, COMSATS Institute of Information Technology Pakistan in 2008.