Impact of Software Design Patterns on the Quality of Software : A Comparative Study

Muhammad Noman Riaz
Department of Computer Science
Virtual University of Pakistan
Lahore, Pakistan
MS150400221@vu.edu.pk

***Abstract* - The quality of software systems depends on several factors and one of them is how the software architects use the design patterns in the design of softwares. This paper comprehensively surveys the existing software design patterns available. The study has two fold objectives. The first and foremost objective is to evaluate the design patterns in terms of their design and quality attributes. The second objective is to provide a summary as how these design patterns affect the software quality. The literature reveals that only four quality attributes of design patterns have been investigated and there is no consensus available on their impact.**

Keywords - Software Quality, Software Design Patterns, Software Maintainability, Level of Design, Quality Attributes

I. INTRODUCTION

The software design patterns are considered as the effective and proven solutions of the recurring design problems. This provides the software architects and the programmers to take advantage of already tested solutions and prevents them in reinventing the optimum solutions and thereby saving a lot of development time [10]. It is also evident from the literature that the reinvented solutions will take more time to be developed and subsequent testing and validation as well as the quality of reinvented solutions at times are not upto the desired standards [10]. This paves the way to the adoption of already proven and tested software design patterns by the practisoners and researchers of software design and development community.

The adoption of design pattern have following advantages [16]: (1) The design patterns have the capacity to improve not only the quality of software but also enhances the productivity of the software developers, (2) The novice programmers or developers experience and skills increase as they study the idea of existing design patterns and try to use them in their current tasks, (3) The design patterns adoption by the experienced and skilled programmers promotes the culture design pattern utilization, (4) The communication between different participants of software development processes gets improved and matured. Nevertheless, no empirical evidence is available that justifies the above discussed advantages.

In order to explore the impact of software design patterns on the quality of software, several literature reviews and experiments have been conducted by the research community as well as by the software practitioners. If the software possesses certain attributes like reliability, reusability, maintainability etc. as per the requirements of the stakeholders then the software can be termed as quality software.

In this paper we have studied the impact of Gang of Four (GoF) design patterns on external quality attributes by combining and comparing the available literature. The second objective of the study is to evaluate the impact of GoF design patterns on the development of software as far as software quality is concerned. This literature survey will help the software designers to choose the best possible design pattern keeping in view the application area. Furthermore, the results obtained from this study will assist the researchers and practisoners in identifying the research gaps between software quality and design pattern.

The section II of the paper discusses the previously held studies that assessed the impact of design patterns on the software quality. The section III of the paper discusses the primary studies from the literature. The section IV summarizes the findings and results of the survey and section V provides the conclusion of the study.

II. LITERATURE SURVEY

In this section we present the state of the art literature survey of the studies that have tried to link the relation between the design patterns and the software quality attributes.

A. Design Pattern and Software Maintainability

In [1] the authors have conducted an experimental study in which they assessed the impact of design patterns on the maintenance of the software systems. During the study they have selected four software systems as subjects of the study to evaluate the impact of design pattern and finally compare them. The five design patterns have been applied on the selected subject systems that include: Composite, Decorator, Visitor, Observer and Abstract Factory. The authors revealed

978-1-5386-1370-2/18/$31.00 ©2018 IEEE

that the use of software pattern is beneficial even for the simple solution but it is not viable to use design pattern in every case. Furthermore, the authors were of the view that it is nor mandatory to use the proven design pattern to find the solution of smaller problem instead the software engineers' common sense should be used in deciding whether the design pattern to be used or not, and if there is a doubt then as a default approach the software designers must opt for the appropriate design pattern.

In [2], the authors have used the same four software systems for the study as that of [1]. In order to increase the fidelity of their study the authors of [2] conducted the study in real programming environment rather than carrying out paper and pen based research. As the results obtained from their study it is not appropriate to generalize the effects or impacts of design pattern on the maintainability of the software system / platform. As per the findings of [2] each design pattern have its own impact on the software maintainability. The design patterns like Decorator and Observer can be easily comprehended by the practisoners with no or little knowledge of the patterns. However, the Composite has shown some issues as it is heavily reliant on recursion. In case of Visitor software design pattern, the understanding is extremely difficult and time consuming as well as the desired amendments will come up with poor level of correctness.

In [3] the authors have performed the replicated experiment based on the original experiment carried out by the authors of [1]. The authors in [3] have used two software systems as subjects instead of using four software systems as used in [1]. Only one statistically important result have been found due to limited scope of the experiment and that is the non – pattern centered version of one system is further sustainable and can be extended easily.

Similarly the authors of [4] conducted the same replicated study of [1]. The two software systems have been used in the study. The authors have used three design patterns that include: Decorator, Composite and Abstract Factory. They found contradicting results because their study concluded that the systems with design patterns contribute in lesser extent in the maintainability of the software systems.

The authors of [5] have conducted the same replication study as that of [1] with same number of software systems as subjects. During the study the authors have maintained focus on four design patterns that include: Composite, Decorator, Visitor and Observer; and only one design pattern is implemented in each software system. As per their findings they conclude that the type of design pattern have no impact on the mainatability and understanding of the software system. In [6] the authors have carried out the same replicated experiment based on the original experiment of [1]. In this experiment the authors have studied three design patterns that include: Abstract Factory, Composite and Decorator. Based on the findings of their experiment they concluded that the pattern based designs not only take more amount of time but also capable of producing large number of faults than its non-pattern based counterpart.

The authors in [7], have discussed the relationship between the design patterns and software system maintainability as far as the understanding and modifiability is concerned. In the study they have used three design patterns that include: Composite, State and Chain of Responsibility. Their study revealed that he diagrams become more difficult to understand and modify when design patterns are used.

The authors of [11] have conducted an experiment in which they studied the effect and impact of Visitor design pattern on both the comprehensibility and maintainability of the software system. During the study the experiments have been performed in which the comprehensibility and maintainability have been studied in terms of eye-fixation and eye-movement. The conclusion of the study reveals that the Visitor design pattern takes a large amount of time in comprehension and maintenance.

The authors in [12] also studied the impact of design patterns on the mainatability of the software system. The authors have thoroughly analyzed 300 revisions of JHotDraw software platform. They concluded that when more instances of design pattern are used it increases the maintainability of software.

B. Software Design Patterns and Evolution

In [14] the authors have studied the changes the patterns are to face and in what manner. They also analyzed what type of changes the design pattern experiences and to what type of changes they are more prone to. During the study they found out that the pattern like Composite Adaptor, Observer, Decorator, Visitor and Factory. Their study revealed that the abovementioned patterns encounter more changes than the design patterns not included in the study.

In [19] the authors have studied the level of proneness the software design pattern is subjected.

During their study they conducted the experiment on different design patterns. They took five different systems for the study. After performing the experiments on these systems the authors found that the patterns are changed. The non-participant classes of the design pattern are more prone to changes than the participant classes of the design patterns.

The authors of [18] have conducted a study on commercially available C# system. In this study the authors have studied the association between software design patterns and proneness to change. The study as that of [19] also found that the participating classes of design patterns experienced more changes than the non-participating classes. The patterns like Visitor, Singleton, Strategy, Proxy, and Method experienced more changes than the patterns like Creator and Command.

C. *Performance and Software Design Patterns*

In [15] the authors have studied the effect of two dissimilar design patterns on the performance of a certain software platform. Two design pattern; Façade and Command have been picked for the study, as they possess similar functionalities. Because of similar functionalities they can be used interchangeably. After performing the test on nine different software platform the study revealed that the Façade pattern is better in performance than the Command pattern.

In [13] the authors have implemented the Stack pattern on the commercially available DSP processor manufactured by Texas Instruments. The study revealed that the more execution time will be consumed by the object oriented platform in case of State design pattern.

D. *Faults and Software Design Patterns*

The authors of [9] have studied the impact of software patters in terms of defect rates on the commercially available large industrial software application programme written in C++. The five design patterns were chosen for the study that includes: Template Method, Abstract Factory, Singleton, Observer and Decorator. During the study the authors revealed that two design patterns, Singleton and Observer, have large code structures and hence needs more consideration. Due to low coupling and high compactness, the Factory design pattern is likely to experience less number of defects.

The authors of [17] have studied the link between the design pattern and fault-proneness on a commercially available software system written in C#. The authors have studied thirteen design patterns and they concluded that the participating classes of design patterns are more fault prone as compare to non – participating classes.

The authors of [8] have conducted a study on the computer games keeping view the impact of design pattern and fault or defect frequency. After studying 11 design patterns they concluded that there is no correlation exists between software defects and the number of design pattern instances. The Observer design pattern have shown positive impact as the number of software defects decreases while the Adopter design pattern have depicted negative impact as the number of defects are quite high.

II. PRIMARY STUDIES REVIEW

During the study we have identified 17 relevant primary studies carried out by different researchers. To meet our research goal we have evaluated and analyzed these studies.

A. *Segregation of Primary Studies with Respect to Quality Attributes*

In table 1 we have presented the relevant primary studies based on quality attributes. Based on these studies we have identified 4 quality attributes.

TABLE 1: DISTRIBUTION OF STUDIES BY S/W QUALITY ATTRIBUTES

S/W Quality Attribute	Studies
Maintainability	[1] [2] [3] [4] [5] [6] [7] [11] [12]
Evolution and Change Proneness	[14] [18] [19]
Performance	[13] [15]
Fault-Proneness	[8] [9] [17]

B. *Segregation of Primary Studies with Respect to Level of Design*

In this research work we have divided the primary studies into two groups. One of these groups have discussed the quality of the software at the design level based on the impact of chosen design pattern. The other group investigated the design pattern impact at the class level.

TABLE II : DISTRIBUTION OF STUDIES BY LEVEL

Level	Studies
Design Level	[1] [2] [3] [4] [5] [6] [7] [8] [9] [11] [12] [13] [14] [15]
Class Level	[17] [18] [19]

C. *Definitions of Quality Attributes Proxy*

The primary studies define each of the quality attributes in some manner. The term proxy is used to quantify the quality related attribute to be computable. The table 3 below depicts the use of proxy in each of the consulted primary study.

TABLE III: QUALITY ATTRIBUTES PROXY DEFINITIONS

Study #	Proxy
[1] [2] [3] [5] [6]	1- Time required for understanding and modification. 2- Correctness.
[4]	Time required for understanding and modification.
[7]	1 - Understandability in terms of time, the # of correct answer to some questions and efficiency. 2 – Modifiability in terms of time, score of modification and efficiency.
[11]	The duration of eye-fixation are used to measure the attention spent to perform comprehension and modification tasks.
[12]	Data collected from the source code and analyzed using probability model.
[14] [19] [18]	# of changes
[15]	# of requests and the processing time of each request
[13]	Memory need and Execution time
[8] [9] [17]	# of faults

D. *Materials and Subjects*

In this sub-section of the study we will discuss the materials and subjects used in the undertaken primary studies. We have divided this sub-section into two parts: The first part describes the materials and subject pertinent to the maintainability experiments and the second part describes the materials and subject used in other primary studies. The primary reason for this division is that the experimental studies involve people while the other studies do not involve people or subjects.

1) *Experiments of Maintainability*

The results obtained from the studies [1] and [2] are strongly evident than the rest of the primary studies because the persons involved in these two studies. The primary studies [2]-[6] are the replication of study [1] and they provide deeper analysis of study [1] through extensive number of experiments performed. The studies [3] – [6] are the studies that were undertaken by the students. In study [12] no subjects were required because the authors have collected the measures and then measured the maintainability accordingly [21].

2) *Other Studies Materials*

In table 5 we have presented the studies in which the researchers have used the materials to study the impact of design pattern on fault, performance and evolution.

TABLE IV: MATERIALS USED IN THE OTHER STUDIES

Study	Materials (Systems)
Evaluation and Change Proneness	
[14]	Three open source systems: JHotDraw, ArgoUML and Eclipse-JDT. Each one of these systems has two releases. The number of classes in each release of each system is as follows: 164 and 489 for the two releases of JHotDraw; 801 and 2373 for the two releases of ArgoUML; 2089 and 6949 for the two releases of Eclipse-JDT.
[18]	WebCSC – a system consisting of 7439 classes and 266k lines of code. The system associated all changes that have been made in one year.
[19]	Five different systems. Three of them are commercial and two of them are open source systems. The commercial software names are hidden and the open sources are Netbeans and jRefactory. Each one of these systems with two versions. The number of classes in each version of each system respectively is as follows: 199 and 227 for the 1st system; 384 and 404 for the 2nd system; 101 and 201 for the 3rd system; 4138 and 7573 for Netbeans; 546 and 699 for jRefactory.
Performance	
[13]	Embedded system implemented in three different approaches: procedural paradigm (C), object oriented paradigm (C++) and object oriented with State design pattern.
[15]	Document version control system. Two version of the system: one with the Façade pattern and one with the command pattern.
Fault-Proneness	
[8]	A set of 97 java open source games with the reported bugs' data. The number of classes in each game ranges between 37 and 1964.
[9]	The system used is SuperOffice CRM5 product and it has 500000 LOC. The system associated with the weekly evolution and maintenance data in a period of 3 years.
[17]	A commercial software system, written in C#. The system consisting of 7439 classes and 266k lines of code. The system associated all changes that have been made in one year.

III. RESULTS

In this section we have presented the summary of the above discussion in that in which manner and extent

the software design pattern impact the software quality attributes.

TABLE V: DESIGN PATTERNS COVERAGE BY THE PRIMARY STUDIES

Pattern type	Quality attribute	Maintainability									Evolution and change-proneness			Performance		Faults		
	Pattern	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[11]	[12]	[14]	[18]	[19]	[13]	[15]	[8]	[9]	[17]
Creational patterns	Abs. Factory	√	√	√	√		√			N/A	√					√	√	
	Builder											√	√					√
	Factory Method											√	√					√
	Prototype															√		
	Singleton											√	√			√	√	√
Structural patterns	Adapter									N/A	√	√	√			√		√
	Bridge																	
	Composite	√	√	√	√	√	√	√			√					√		
	Decorator	√	√	√	√	√	√				√					√	√	
	Facade														√			
	Flyweight																	
	Proxy											√	√			√		√
Behavioral Patterns	Chain of Resp.							√		N/A								
	Command										√	√	√		√			√
	Interpreter																	
	Iterator											√	√					√
	Mediator																	
	Memento																	
	Observer	√	√			√					√					√	√	
	State							√				√	√	√		√		√
	Strategy											√	√			√		√
	Template															√	√	
	Visitor	√	√			√			√		√	√	√					√

V. CONCLUSION

From the study we concluded that the impact of software design patterns must be further discussed in deeper details and each design pattern should be given adequate attention. From the study it is concluded that the impact of design patterns on evolution, proneness and maintainability is negative, in general. As far as the performance is concerned the number of studies that addressed performance makes it difficult to draw a conclusion. Also, for fault proneness, the results are different from one study to another thus it becomes difficult to make a decision regarding their impact.

REFERENCES

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

[2] Vokáč, M.; Walter, T.; Dag, L.K.S.; Erik, A.; Magne, A.; , "A controlled experiment comparing the maintainability of programs designed with and without design patterns—a replication in a real programming environment." Empirical Software Engineering 9, no. 3: 149-195. 2004.

[3] Prechelt, L.; Liesenberg, M.; , "Design Patterns in Software Maintenance: An Experiment Replication at Freie Universität Berlin," Second International Workshop on Replication in Empirical SoftwareEngineering Research (RESER), vol., no., pp.1-6, 21-21 Sept. 2011.

[4] Juristo, N.; Vegas, S.; , "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), vol., no., pp.7-14, 21-21 Sept. 2011.

[5] Nanthaamornphong, A.; Carver, J. C.; , "Design Patterns in Software Maintenance: An Experiment Replication at University of Alabama," Second International Workshop on Replication in Empirical Software Engineering Research (RESER), vol., no., pp.15-24, 21-21 Sept. 2011.

[6] Krein, J. L.; Pratt, L. J.; Swenson, A. B.; MacLean, A.C.; Knutson, C. D.; Eggett, D.L.; , "Design Patterns in Software Maintenance: An Experiment Replication at Brigham Young University," Second International Workshop on Replication in Empirical Software Engineering Research (RESER), vol., no., pp.25-34, 21-21 Sept. 2011.

[7] Garzás, J.; Félix, G.; Mario P.; , "Do rules and patterns affect design maintainability?." Journal of Computer Science and Technology 24, no. 2: 262-272. 2009.

[8] Ampatzoglou, A.; Apostolos, K.; Elvira-Maria, A.; Antonis, G.; Fragkiskos, C.; Ioannis, S.; , "An empirical investigation on the impact of design pattern application on computer game defects." In Proceedings of the 15th International Academic MindTrek Conference: Envisioning Future Media Environments, pp. 214-221. ACM, 2011.

[9]Vokac, M.; , "Defect frequency and design patterns: an empirical study of industrial code," IEEE Transactions on Software Engineering, vol.30, no.12, pp. 904- 917, Dec. 2004.

[10] Gamma, E. ; Helm, R.; Johnson, R.; Vlissides, J.; , "Design Patterns: Elements of Reusable Object-Oriented Software." Reading, Mass.: Addison-Wesley, 1995.

[11] Jeanmart, S.; Gueheneuc, Y.-G.; Sahraoui, H.; Habra, N.; , "Impact of the visitor pattern on program comprehension and maintenance," 3rd International Symposium on Empirical Software Engineering and Measurement ESEM , vol., no., pp.69-78, 15-16 Oct. 2009.

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

[13] Afacan, T.; , "State Design Pattern Implementation of a DSP processor: A case study of TMS5416C," 6th IEEE International Symposium on Industrial Embedded Systems (SIES), vol., no., pp.67-70, 15-17 June 2011.

[14] Aversano, L.; Gerardo, C.; Luigi, C.; Concettina, D. G.; Massimiliano, D. P.; , "An empirical study on the evolution of design patterns." In Proceedings of the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT . symposium on The foundations of software engineering, pp. 385-394. ACM, 2007. Rudzki, J.; , "How design patterns affect application

performance–a case of a multi-tier J2EE application." Scientific Engineering of Distributed Java Applications: 12-23. 2005.
[16] Prechelt, L.; Unger-Lamprecht, B.; Philippsen, M.; Tichy, W. F.; "Two controlled experiments assessing the usefulness of design pattern documentation in program maintenance," IEEE Transactions on Software Engineering, vol.28, no.6, pp.595-606, Jun 2002.
[17] Gatrell, M.; Counsell, S.; , "Design patterns and fault-proneness a study of commercial C# software," Fifth International Conference on Research Challenges in Information Science (RCIS), vol., no., pp.1-8, 19-21 May 2011.
[18] Gatrell, M.; Counsell, S.; Hall, T.; , "Design Patterns and Change Proneness: A Replication Using Proprietary C# Software," 16th Working Conference on Reverse Engineering, vol., no., pp.160-164, 13-16 Oct. 2009.
[19] Bieman, J.M.; Straw, G.; Wang, H.; Munger, P.W.;Alexander, R.T.; , "Design patterns and change proneness: an examination of five evolving systems," Ninth International Software Metrics Symposium, vol., no., pp. 40- 49, 3-5 Sept. 2003.
[20] IEEE Std.1061, Std. 1061 for a Software Quality Metrics Methodology. New York: Institute of Electrical and Electronics Engineers.Procedia Computer ScienceVolume 109, 2017, Pages 521-528.