A RESEARCH STUDY ON THE CHOICE OF DESIGN PATTERNS

Pedro Mauricio Tavares, Dublin, Ireland

***Abstract*—Design patterns address problems that occur repeatedly in software development, describing a solution to a problem, in such a way that the solution can be reused. There is evidence that patterns can be beneficial in software, but can cause problems if applied inappropriately, therefore, developers should have a good understanding of how to use, when not to use and good guidance of where to apply design patterns. Some patterns have advantages over others in terms of flexibility or learning, others are suitable for certain situations. This paper represents a study of the relationship between design patterns and software quality attributes, with the main intent to serve as a guideline to make better decisions of when to apply the right patterns on applications during software development or refactoring.**

***Index Terms*—Design Patterns, Software Defects, Quality, Maintainability**

I. Introduction

THE idea of design patterns was acquired from Christopher Alexander, who suggested that certain common patterns of building design were effective [15], these concepts have since been accepted in software engineering. Design patterns are true object-oriented software design solutions to solve design problems.
According to Gamma in "Gang of Four" book, which we will refer to as GoF in this paper [10], the use of design patterns provides several advantages, from increased reusability and improved maintainability to comprehensibility of existing systems.

If applied properly, design patterns can lead to better quality and maintainable software. However, you need to be aware that for each design problem encountered, an associated pattern can be used, thus, many developers and engineers, even the most experienced, find them difficult to apply. Patterns are beneficial, but they also have drawbacks, engineers need experience to apply them effectively, contrarily, it might lead to problems on your application.

The purpose of this research is to present how design patterns affect applications if applied incorrectly and present guidelines to apply effectively to avoid problems. The remainder of this paper is organized as follows. Section 2 presents the methodology used to elaborate this paper. Section 3 gives a brief overview of design patterns. Section 4 shows how using patterns and frameworks help to improve software development. Section 5 and 6 evaluates patterns on quality attributes, demonstrating user's perception and how patterns should be applied. Section 7 shows gaps to our approach. The concluding remarks and future work are discussed in Section 8.

II. METHODOLOGY

This paper was elaborated from case studies and literature review of five (5) journals which includes several authors. After analysing them, we noted that there were more related and informative journals and books, which were also included in this paper. Figure 1.0 shows the main papers used as core reference of this project, nevertheless throughout this document we also included important information from other relevant sources.

People are considered the weak link in an Information Security System, Social Engineering is the art of exploiting such a link Kowalski. Social Engineering is "people hacking" and involves maliciously exploiting the trusting nature of human beings.

Alghamdi & Qureshi, 2014	Impact of Design Patterns on Software Maintainability
Ali & Elish, 2013	A Comparative Literature Survey of Design Patterns Impact on Software Quality
Ampatzoglou et al., 2011	An empirical investigation on the impact of design pattern application on computer game defects
Edwin, 2014	Software Frameworks, Architectural and Design Patterns
(Zhang & Budgen, 2013)	A survey of experienced user perceptions about software design patterns.

Fig. 1
Core Documents Source

III. OVERVIEW OF DESIGN PATTERNS

Patterns have impacted on object-oriented software design, whereas being tested solutions for common problems, they have become a vocabulary for talking about a design [15]. For example, consider where a developer wishes to reuse two incompatible classes (A and B), A sends two parameters in a message, but B accepts only one parameter. If a developer changes B, this could create several problems affecting other classes. Instead, we can create a different class C, which can receive messages from A, which has two parameters and send one parameter to B. This is an example of a design pattern called Adapter [9]. Design patterns can be categorized in the following way:

Creational patterns: Aim to solve design problems by creating objects.
Structural patterns: Aim to solve problems by providing a simple model for relationships between entities.
Behavioural patterns: Address problems by trying to identify common communication in an effective way between objects.

In the following sections we present a literature research to provide software engineers a better understanding of

how to choose and apply patterns.

IV. USE OF FRAMEWORKS AND DESIGN PATTERNS

Frameworks are examples of design patterns used appropriately [8], to achieve flexible and less error prone applications in a more effective way. Additionally, frameworks and design patterns are cheaper and easier to deliver a product compared to other conventional methodologies. However, Edwin points that development time might increase unexpectantly as developers need time to learn the framework they intend to use. This is common when a new or unknown framework is first encountered by a Software Engineer. Furthermore, developers must work to prevent a phenomenon called "code-bloat" where if frameworks are applied incorrectly it can lead to performance loss and unnecessary API functions.

The learning curve of Frameworks highlighted by Edwin is a valid argument, but this can be applied for any technology, therefore learning is something that became normal and necessary in the technological field. Moreover, the learning process to effectively use a specific framework will be lighter than doing everything, including applying design patterns, from scratch and once the developer learns, it can be re-used when necessary. Thus, the correct use of existing Frameworks is advisable and recommended whenever possible.

V. EVALUATION OF DESIGN PATTERNS

Many studies suggest that design patterns make a difference in software quality and maintenance [2]. This section shows how to evaluate patterns for quality and maintainability, so that you will have a better judgment when applying them.

A. DESIGN PATTERNS ON QUALITY AND MAINTAINABILITY

Design patterns are a factor in influencing software quality. Many experiments and studies have been conducted to analyse this influence. When defining quality, the software needs to possess certain attributes such as maintainability, reliability and fault-proneness [11]. A study conducted [5] showed that there is no consensus among studies regarding the impact on quality attributes, whilst some studies reported positive or neutral, others reported negative impact. There were 4 quality attributes covered in the study: maintainability, evolution, performance and fault-proneness. For performance, the number of studies addressed, and the number of covered patterns make it hard to draw a conclusion. For fault proneness, the result varies from one to another, making it difficult to know the impact.

One of the problems with the results is that, although there seems to be a fair coverage, not every single pattern's impact was evaluated over every quality attribute. For instance, on impact on maintainability, only 6 patterns were covered and only three for performance. As Ali & Elish labelled, the impact of patterns on software quality needs to be evaluated in-depth, considering a wider range of patterns, more importantly, we must consider and prioritize other quality attributes to avoid conflicts and don't "blame" design patterns for poor quality as some implementations of patterns might result in a more flexible and expandable architecture, but could also increase complexity. The decision about the use of a pattern for quality is to some degree a matter of subjective judgement, therefore, before determining the use of patterns to improve quality, we first must decide what the successful critical factors are [17].

For software maintainability, different researchers have different conclusions, for that a tool was proposed [4] to measure the efficiency of patterns on maintainability followed by a survey to evaluate the tool. The results show that the tool provides a good way to evaluate patterns. Most responders agreed it helps choosing a design pattern to produce more maintainable systems. However, we think that the results can't be considered a final guidance on the choice as we cannot know the responders' expertise. The effect of patterns on the maintainability is governed by different factors such as size, developer's expertise [7], all these should be analysed by software engineers.

B. DESIGN PATTERNS FOR GAMES SOFTWARE

Another field where patterns are applied is software games, to make a better choice of patterns, it is worth to have a guidance on defects and what could be caused by improper use.

An empirical study on ninety-seven (97) Java open source games demonstrated that there is a correlation between design patterns and games software defects [6]. The results suggest the following: The overall number of pattern instances is not correlated to defect frequency, some design patterns appear to have a significant impact on the number of bugs and debugging rate, additionally, some patterns have certain characteristics that influence defect frequency somewhat. Particularly, if the number of instances of these patterns increases, the number of open bugs in a project decrease. Patterns correlated to defect frequency negatively are: Abstract Factory, Singleton, Composite, Observer, State, and Strategy.

On the other hand, patterns that appear to influence bug fixing rate are Singleton, Adapter, Observer and Decorator. Singleton and Decorator appears to be negatively correlated to debugging efficiency as we remove these pattern instances, more bugs are fixed. Adapter pattern instances also seems to hinder bug fixing activities. Nevertheless, Observer pattern enhances debugging procedure as it pro-

vides a structured way of managing interaction among application layers, such as user interfaces, and game logic. This study provides a guideline to investigate and apply design patterns on software games, however, further studies should be considered to provide more accurate answers. Also, the developer's expertise that conducted the bug fixing wasn't evaluated, so we can't know if other patterns could have been used.

VI. USER PERCEPTION

A mapping study was conducted by Zhang & Budgen [19] to examine an empirical knowledge for software design patterns. There was mixed evidence about the conclusion of the patterns studied, thus patterns must be accessed singly to determine their proper use.

To analyse patterns as single, a survey was conducted by [18] to identity what patterns are likely to be used. A total of 206 users participated in the study.

Degree	Primary Role			
	Developer	Researcher	Teacher	Student
Associate	1	0	0	0
Batchelor's	30	1	0	2
Masters	31	8	2	4
PhD	16	70	37	1
Other	3	0	0	0
Total	81	79	39	7

Fig. 2
Core Documents Source

The study concluded that Observer, Composite and Visitor patterns stood out among the responders for positive evaluation, while Prototype, Flyweight, Interpreter and Memento obtained a negative rate. Singleton pattern was considered the best known, the conclusion about whether it could be valuable for a software application is mixed, especially among more experienced developers. Below, we present a guidance of when to apply and use some patterns.

A. VISITOR PATTERN

According to most users, Visitor Pattern can make an application more maintainable, reducing coupling which makes it more extensible. Nevertheless, novice developers need to be careful about its complexity, which is the main cause of this pattern been misused by many people, leading to hard maintainable code [18].

B. COMPOSITE

Composite Pattern makes instances of classes behave in the same way as a group of objects, which facilitates the inclusion of new objects. For this reason, this pattern was considered to have a positive impact on software. Additionally, it facilitates the creation of generic data models, making the system more flexible, however, many objects may be referenced since this pattern doesn't restrict the type during instantiation [12].

C. OBSERVER

Observer pattern is not correlated to software bugs, if the number of instances increases, the number of software bugs tend to decrease. Such a situation happens because the use of observer makes interaction among application layers simpler, consequently, the higher the number of classes that participate in the observer, the higher the debugging efficiency [6]. Additionally, Observer is known for addressing encountered problems and reduces coupling [18].

D. SINGLETON

Singleton must be used carefully, in fact, experienced software developers are in favour of not using it for the same reasons that other studies suggest (Ampatzoglou et al., 2011), where Singleton pattern is like global variables, in the sense that the instance returned is accessible from many objects and changes may have a large impact across the system. If this pattern is used carelessly, global variables might be added to the system, which is bad practice [13] [16]. An example where this pattern can be used safely is on logging classes.

E. ABSTRACT FACTORY

One of most recommend [18], this pattern decouples the creation of an object from its use. The implementation makes the system more expandable and improve reusability, thus, it is suitable if these are the main requirements of a system. Nevertheless, although this pattern is considered to have a positive impact, it's advised to not overuse it, because it can induce to unnecessary complexity, affecting understandability of the system [12].

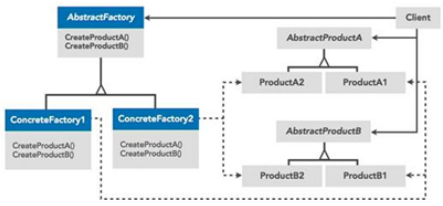

Fig. 3
Abstract Factory design [1]

VII. GAPS AND CRITICISM

While this paper presented numerous studies on design patterns and shows different situations of when to apply them, only a few of the 23 patterns of GoF book were covered, therefore, the applicability of these patterns can be objected, a software engineer could make a better judgment if they had a solid understanding of all patterns. Additionally, no mechanism to validate the background of the responders involved in the surveys was provided, which makes the results questionable.

[1] Source: https://www.lynda.com

Another major point for our work was the fact that we did not expand the case study to add other software quality attributes such as scalability, learnability etc as well as research and analyse studies such as what design patterns are suitable for novices [1]. Lastly, subsequent studies to verify if the pattern choices ended up being corrected were not conducted, therefore, we cannot fully guarantee that the results are 100% accurate.

VIII. CONCLUSION AND FUTURE WORK

In this study, we investigate the relationship of design patterns on software applications and how they should be applicable to improve software development in an effective manner. Several studies were analysed, each of them part of a different domain.

Firstly, we did an overview of design patterns, then we presented a theory of their use in frameworks and how it can facilitate development lifecycle and build more maintainable code. We also presented some studies of how design patterns affect software quality and maintainability. A more detailed usability of some of the researched patterns was included to provide users with more knowledge of when and where to apply design patterns such as Singleton, Observer, Composite, etc.

Future studies can widen the range of patterns. Different quality metrics can be added, it would be interesting to investigate how design patterns affect learnability for instance. Furthermore, the domain can be expanded to Artificial Intelligence or Security Systems. Additionally, since software complexity is related to quality and maintainability, another worthwhile topic to include could be the relation of design patterns with complex systems [**?**].

References

[1] Abdul Jalil, M., & Noah, S. (2007). The Difficulties of Using Design Patterns among Novices: An Exploratory Study. 2007 International Conference On Computational Science And Its Applications (ICCSA 2007), 97-103. doi: 10.1109/iccsa.2007.32

[2] Ahmad, N., & Waqas Boota, M. (2014). Evaluation Amid different Software Design Patterns. International Journal Of Computer Applications (0975 – 8887), 105(11), 28-34. Retrieved from https://www.semanticscholar.org/paper/Evaluation-Amid-different-Software-Design-Patterns-Ahmad-Boota/71c13284189925f114cd9cfcd533dc6a9d203126

[3] Alexander, C. (1980). The timeless way of building (1st ed.). New York: Oxford University Press.

[4] Alghamdi, F., & Qureshi, M. (2014). Impact of Design Patterns on Software Maintainability. International Journal Of Intelligent Systems And Applications, 6(10), 41-46. doi: 10.5815/ijisa.2014.10.06

[5] Ali, M., & Elish, M. (2013). A Comparative Literature Survey of Design Patterns Impact on Software Quality. 2013 International Conference On Information Science And Applications (ICISA), 1, 1-7. doi: 10.1109/icisa.2013.6579460

[6] Ampatzoglou, A., Kritikos, A., Arvanitou, E., Gortzis, A., Chatziasimidis, F., & Stamelos, I. (2011). An empirical investigation on the impact of design pattern application on computer game defects. Proceedings Of The 15Th International Academic Mindtrek Conference On Envisioning Future Media Environments - Mindtrek '11. doi: 10.1145/2181037.2181074

[7] Ampatzoglou, A., Frantzeskou, G., & Stamelos, I. (2012). A methodology to assess the impact of design patterns on software quality. Information And Software Technology, 54(4), 331-346. doi: 10.1016/j.infsof.2011.10.006

[8] Edwin, N. (2014). Software Frameworks, Architectural and Design Patterns. Journal Of Software Engineering And Applications, 07(08), 670-678. doi:10.4236/jsea.2014.78061

[9] Freeman, E., Robson, E., Sierra, K., & Bates, B. (2004). Head First design patterns(1st ed., pp. 243-283). USA: O'Reilly Media, Inc.

[10] Gamma, E., Johnson, R., Vlissides, J., & Booch, G. (1994). Design Patterns: Elements of Reusable Object-Oriented Software (1st ed.). New Jersey: Addison-Wesley Professional.

[11] Jinzenji, K., Hoshino, T., Williams, L., & Takahashi, K. (2013). An experience report for software quality evaluation in highly iterative development methodology using traditional metrics. 2013 IEEE 24Th International Symposium On Software Reliability Engineering (ISSRE). doi: 10.1109/issre.2013.6698884

[12] Khomh, F., & Gueheneuce, Y. (2008). Do Design Patterns Impact Software Quality Positively?. 2008 12Th European Conference On Software Maintenance And Reengineering, 274-277. doi: 10.1109/csmr.2008.4493325

[13] Koopman, P. (2010). Better embedded system software (pp. 186-197). [S. l.]: Drumnadrochit Education.

[14] Pressman, R. (2010). Software Engineering: a practitioner's approach (7th ed.). New York: McGraw-Hill.

[15] Sommerville, I. (2016). Software engineering (10th ed., pp. 209-212). Harlow: Pearson Education.

[16] Sward, R., & Chamillard, A. (2004). Re-engineering global variables in Ada. ACM Sigada Ada Letters, XXIV(4), 29-34. doi: 10.1145/1046191.1032303

[17] Wendorff, P. (2001). Assessment of Design Patterns during Software Reengineering: Lessons Learned from a Large Commercial Project. Proceeding CSMR '01 Proceedings Of The Fifth European Conference On Software Maintenance And Reengineering, 77-84. doi: 0-7695-1028-0

[18] Zhang, C., & Budgen, D. (2013). A survey of experienced user perceptions about software design patterns. Information And Software Technology, 55(5), 822-835. doi: 10.1016/j.infsof.2012.11.003

[19] Zhang, C., & Budgen, D. (2012). What Do We Know about the Effectiveness of Software Design Patterns?. IEEE Transactions On Software Engineering, 38(5), 1213-1231. doi: 10.1109/tse.2011.79