Available online at: https://ijact.in

Date of Submission	28/07/2020
Date of Acceptance	20/08/2020
Date of Publication	05/10/2020
Page numbers	3841-3847 (7 Pages)

ISSN:2320-0790

COMPUSOFT
An International Journal of Advanced Computer Technology

AN EMPIRICALLY TESTED DESIGN PATTERN SELECTION FRAMEWORK BASED ON DEVELOPER EXPERIENCE

Eddy Khonica[1], ImranMedi[2]and Muhammad Ehsan Rana[3]

[1,2,3] Asia Pacific University of Technology and Innovation, Kuala Lumpur, Malaysia
[1]eddy.khonica@gmail.com,[2]Imran.medi@apu.edu.my, [3]muhd_ehsanrana@apu.edu.my

Abstract: In most IT projects, software maintenance had always been a difficult task to perform especially when the software is not designed properly. This has led to numerous changes all over the places despite a minor change request. Design patterns are widely known for its proven solution to recurring problems. If fully utilized, design patterns can prevent such incident from happening as changes can often be added without affecting existing components. However, due to numerous design patterns that are available these days, it is difficult to select the right pattern. In addition, existing studies still lack a proven guideline to select the right pattern. This difficulty is faced by many beginners and experienced developers. The aim of this research is to propose a framework to select the most suitable design patterns in software development. The research methodology used is quantitative research where primary data are collected using survey and questionnaires. This is mainly to find out developers' attitudes towards the use of design patterns. 25 out of 48 papers regarding design patterns are referenced in this paper. The framework developed is evaluated and empirically tested using expert judgement and controlled experiment. The result shows that the framework does help in evaluating and selecting the most suitable pattern. Therefore, the authors highly recommend developers to make use of this framework in software development.

Keywords: Design Patterns, Software Flexibility & Reusability, Developer Experience, Framework, System Architecture

I. INTRODUCTION

A poorly designed software would eventually suffer when it need to accommodate new changes. In the long run, it becomes way too expensive to perform maintenance. Reference [1] produced a chaos report stating that only 16.2% projects related to IT are completed within the allocated budget and time. Furthermore, the costs to maintain software are rising greatly in which maintenance phase is estimated to take 90% of project cost[2]. Fig. 1 illustrates the fluctuation of maintenance cost from 1970 to 2014.

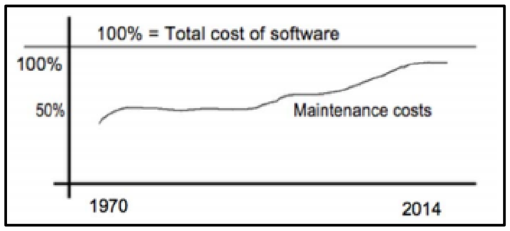

Fig. 1. The Evolution of Software Maintenance Cost [3]

There is an implicit relationship between project failure and software maintenance as project will eventually fail when the maintenance cost exceeds the total project cost. Improper software design could be the cause of this issue. Relating this to the difficulty in selecting the right design pattern makes this research significant.

Reference [4]defined design pattern as *"a tested solution to a standard programming problem"*. It is widely used in most object-oriented software projects as it provides successful solution in solving recurring problem[5], [6]. Due to the large number of design patterns available these days, it is very difficult to select the right pattern in software development[7]. Furthermore, inappropriate use of pattern will make system more complicated[8], [9]. Therefore, it is extremely important to select the right pattern. The motivation behind this research is to come up with a framework that aids developers in selecting the right design patterns.

The following are the research questions formulated for this research:

RQ1 How to ensure correct selection of design patterns?

RQ2 What are the actions that can be taken to improve developers' knowledge on design patterns?

RQ3 How design patterns can be used to promote software flexibility and reusability?

RQ4 What makes a good design pattern selection framework?

The output of this research is a design pattern selection framework which aids in selecting the right design pattern. There is other framework that helps in selecting design pattern. Reference [7] comes up with an automated framework which stores the characteristics of every pattern into a repository and then match it with the problems to suggest the right pattern. The framework developed in this research is different from the one mentioned above as it focuses more on the evaluation on the need, selection, and implementation of pattern. In addition, it emphasizes on the developers' years of experience in addition to the characteristics of the patterns. This is because the software development is all about the experiences.

The next section provides the literature review on why design patterns are not being fully utilized in software development, followed by the research methodology used in this research. Then the next section represents result and discussion which contain the data analysis and proposed framework. Lastly it concludes this research with recommendation and future work.

II. Literature Review

A. *Selection of Design Patterns*

Selecting which design pattern to be used is a common difficulty faced by many beginners and experienced developers [7], [8]. One possible reason is that there are similar patterns that can be used interchangeably. Reference [10] highlighted that existing studies on design patterns are not enough to provide a guideline on when to and not to use a pattern. This shows that there is no empirical evidence whether design patterns should or should not be applied.

Design patterns must not be mistaken as a global solution which lead to better software development [11]. It is a set of practically tested solutions to problems with known pattern [12]. A conclusion can be drawn from these two statements in which there is no design pattern that could solve all types of problem faced in software development, but there are patterns that can be used for certain known problem. It all depends on the nature of the problem. However, there are many things that could be done to find the right pattern to be used. Reference [13] proposed a tool in respect to maintainability evaluation which can help developers to find a suitable pattern which improves system maintainability. However, no other researchers have used the proposed tool. Therefore, there is no solid proof on how the tool could aid developers in selecting the right pattern.

Furthermore, by studying the interrelation between one design pattern to another, it may lead developers to the most suitable pattern [8]. Therefore, developers must continue to learn and explore design patterns as well as identifying their relationships between one another. This will help them in identifying the most suitable design patterns to be used. It must be emphasized that the selection of design patterns must not be made for the sake of having a design pattern. It highly relies on the applicability and suitability of the pattern to the problem encountered. When design patterns are not applied correctly, it poses various disadvantages to the software development process [8]. Wrongly used design pattern causes the design to be more complicated as well as making maintenance harder [9]. This indicates that the selection of design patterns must be done with care. Thus, it is extremely important to select the right pattern.

B. *Learning Curve and Applicability*

Design patterns have significantly affected how a software is being developed in which it provides the opportunity for novices to learn and rises their contribution to the project [14]. Reference [14] highlighted that during their time, it was difficult to compare patterns due to the unavailability of measurement metrics. Furthermore, there were no ample opportunity for developers to learn and apply these patterns in practice. However, over the years, other researchers have been comparing these patterns to check its applicability in the software development [12], [15]. Hence, metrics are being used to calculate the design patterns impact on various quality factors (e.g. maintainability, flexibility, etc.). Some of the research gaps have been filled from time to time, but there is no solid proof on how design patterns help novices to learn. Although design patterns appear to be useful in software development, it does not really help beginners to learn how to design software [10], [12]. As opposed to [14] statement where design patterns provide opportunities for novices to learn; Reference [10], [12] claims that beginners do not really get to learn software design through the use of design patterns. These contradictory statements show that experts' views on design patterns may change from time to time.

Depending on the experiences of the users rather than their role, only the functions of the design patterns can be optimum [11]. It is of utmost importance for developers to rely on their experiences in selecting and applying design patterns in software development due to the complexity of object-oriented design [8], [16]. Reference [8], [11] both highlighted that the skillset and experiences on how to apply the design patterns are more important than the concept itself. This could be a reason why developers are not fully utilizing design patterns as they are not experienced enough to use the design patterns correctly. Improper implementation of design pattern will complicate the design which makes maintenance harder [9]. Thus, developers are encouraged to learn and practice design patterns continuously.

A practical way to learn how to implement design patterns is to refer to the sample codes provided for each pattern [8]. Reference [17] has clearly documented each pattern as well as providing code snippets on how the pattern can be implemented. This is a good start where developers can directly see the implementation from class level hierarchy, thus allowing them to gain a better understanding on the patterns. Furthermore, a method to better understand a system architecture is to locate design patterns instances in the source code [12], [18]. Lastly, it is extremely crucial for practitioners to attend training on design patterns to improve their knowledge and coding abilities [19]. These are a few ways to help developers in improving their skills on software design.

C. Should Design Patterns be Used

The lack of clear evidence to suggest that design patterns improve software quality has been a recurring topic in the literature for close to 20 years. For example, in 2001, [20], [21] highlighted that the use of design patterns does not necessarily result in a better design. Seven years later, [22] also claimed that design patterns may reduce software quality if it not applied carefully. Furthermore, there is little proof on how design patterns could improve the quality of the product [23]. Not only do design patterns not improve software quality, but it may also increase the complexity of the software [8]. Finally, it is again emphasized that design patterns do not always improve software quality [24]. For this reason, it would be understandable if developers decide not to use design patterns.
All the above researchers have shown the negativity of using design patterns in software development in which it does not guarantee to improve software quality. However, design patterns also have its own benefits which must not be overlooked. For instance, design patterns ease system modeling which help developers in gaining a better understanding of the system, thus eases maintenance [18], [25]. It also incorporates flexibility and maintainability into the developed system [11]. By taking flexibility into account, the system quality and reliability can be improved within the constraint of time and budget [26]. Furthermore, [6] emphasizes that design patterns allow developers to create a cleaner design which promotes reusability as well as increasing coding efficiency when implemented correctly. The use of design patterns also helps in preventing issues that can lead to major problems, improving code readability, as well as speeding up software development [8]. Last but not least, the use of design patterns also help to cover the lack of documentation, thus allowing developers to understand the system architecture faster [5]. These are the benefits gained from using design pattern. In conclusion, design patterns are encouraged to be used especially when the developers are already fluent with it, so that it can be selected and implemented correctly.

D. Implication of Design Patterns on Source Code

Not only can design patterns be used for system design, but it also improves existing system through refactoring [17]. The refactoring here refers to the activity to identify and reorganize source code based on the most suitable patterns. Reducing code smell is a benefit gained from applying design patterns. There is lesser code smell detected for classes that participate in design pattern[12]. In exchange of the reduced code smell, this does introduce more classes to the system design. It raises the difficulty to understand and maintain the system if it is not documented properly. Reference [27] highlighted that design patterns that are well-documented will improve code comprehensibility. However, the source code will get bloated as the number of lines of codes including comments are going to significantly increase [27], [28]. This is one concern on why design patterns are not being fully utilized as developers will always go for writing the least amount of codes.

E. Disadvantages of Design Patterns

Despite the benefits design patterns provided in system design, it still poses as a difficulty for designers to understand them[25]. This is to be expected as it requires a long time (few years) to master design patterns[8]. Furthermore, design patterns do not necessarily improve software quality and it may increase the complexity of the software[8]. Moreover, it is difficult to conclude whether design patterns impact performance positively or negatively due to the limited number of studies on this area[29]. Meaning that there is no solid proof that performance can be improved by using design patterns. Thus, performance cannot be used to determine if design patterns should be applied.

TABLE I. IMPACT OF DESIGN PATTERNS ON PARTICIPATING CLASSES [27]

	DP Classes	**NoDP Classes**
Greater line of codes	✓	
Greater line of comments	✓	
Lower level of cohesion	✓	
Higher level of coupling	✓	
Higher level of complexity	✓	

Table I shows a few disadvantages for involved classes in design patterns tabulated[27]. Reference [27] highlighted that classes that are using design patterns are characterized by lower level of cohesion and higher level of coupling. Furthermore, classes involved in design pattern have higher complexity than non-participating classes[27]. This may be true as the number of classes is bound to increase by applying design patterns, thus increasing the level of complexity. All these disadvantages could be the reason why developers do not fully utilize design patterns in addition to the difficulty in selecting the right pattern. Table II summarizes the design pattern issues from the papers referenced in this research.

TABLE II. SUMMARY OF DESIGN PATTERN ISSUES

Authors	Design Pattern Issue
[10], [12]	Design patterns do not really help beginners to learn how to design software.
[23], [24]	Design patterns do not always improve software quality.
[30]	The use of inappropriate design pattern will negatively affect the system quality.
[7], [8]	Selecting which design pattern to be used is a common difficulty faced by many beginners and experienced developers.
[27], [28]	Design patterns will increase the number of lines of codes and comments which increases complexity.
[8]	Design patterns do not necessarily improve software quality and it may increase the complexity of the software.
[27]	Lower level of cohesion and higher level of coupling are the characteristics of design pattern classes. Design pattern classes have higher complexity than non-participating classes.
[24]	Excessive number of design patterns used will increase the difficulty in understanding the structure of the design.
[9]	Incorrect implementation of design pattern will make the design more complicated, thus making maintenance harder. Defect rate will be higher when more than one design patterns involved in the implementation of concern.

F. Existing Framework to Select Design Pattern

This section highlights an existing framework that helps in selecting suitable design pattern as well as its weaknesses. Reference [7] proposed an automated framework which stores the characteristics of every design pattern into a repository and then match it with the problems encountered to suggest a suitable pattern. There are some weaknesses that must be highlighted regarding this existing framework. Firstly, it does not really take developers' experience into serious consideration. This is crucial as software development is all about experiences. Furthermore, this existing framework has not been fully proven as no other researchers have utilized it in their research meaning that there is no solid proof that this existing framework has helped developers in selecting the right pattern. Therefore, this research aims to propose a new framework which focuses on the experiences of the developers in addition to the characteristics of design patterns.

III. Methodology

The GoF design patterns are introduced in 1994 and many studies have been conducted ever since. The years of study regarding design patterns considered in this research is from 1994 to most recent. This is to compare experts claims from 20 years ago up till today because experts' point of view may change from time to time. However, most of the references in this research are from 2010 onwards. This is to ensure that the literature conducted is up to date.

This research uses quantitative research approach as it is more accurate to empirically analyze the data collection on many aspects related to developers' views and opinions on design patterns. Conducting survey is the research method used with questionnaires being the research technique used for the data collection. The questionnaires are created using Google Forms and distributed online using professional networking site (e.g. LinkedIn), social media (e.g. Facebook), and online messenger (e.g. WhatsApp). The target survey populations are experienced software developers who have ever used GoF design patterns. In addition, the data collected will be analyzed using correlation in MS Excel to identify the relationships between the research variables. For this reason, the sample size is kept at a minimum of 50 respondents to produce a more reliable result. The survey is conducted for 25 days and there are 55 respondents in total. The findings obtained from the data analysis along with the literature review serve as inputs to the development of the proposed framework. Two types of evaluation are conducted to empirically evaluate the validity of the framework such as expert judgement and controlled experiment. The purpose of these evaluations is to gather feedback to further improve the framework.

IV. Results

4.1 Data Analysis

This section presents the findings of the data collection conducted by survey and questionnaires.

*a) **How Years of Experience Impact GoF Design Patterns***

Table III shows the Pearson's r coefficient between the two variables at 0.807357825. This value indicates that there is a strong relationship between the two as the increase in years of experience will enable oneself to be more familiar with design patterns.

TABLE III. Correlation between Years of Experience and Familiarity with No. of Design Patterns

	Category of Experience	Category of No. of Patterns
Category of Experience	1	0.807357825
Category of No. of Patterns	0.807357825	1

TABLE IV. Correlation between Years of Experience and Design Patterns Usage

	Category of Experience	Frequency of Usage
Category of Experience	1	0.573517656
Frequency of Usage	0.573517656	1

Table IV shows the Pearson's r coefficient between years of experience and frequency of design patterns usage at 0.573517656. This value indicates that there is a partially strong relationship between the two as the increase in years of experience will increase the usage of design patterns. On the other hand, Table V shows the Pearson's r coefficient between years of experience and usefulness of design patterns at 0.356873058. This value indicates that there is a weak relationship between the two as beginner and experienced developers believe that design patterns are useful in software development.

TABLE V. CORRELATION BETWEEN YEARS OF EXPERIENCE AND DESIGN PATTERNS USEFULNESS

	Category of Experience	Usefulness
Category of Experience	1	0.356873058
Usefulness	0.356873058	1

b) How Years of Experience Impact the Way Design Patterns are Applied

Table VI shows the correlation value of 0.661712039 between years of experience and tailoring design patterns. This value indicates that the higher the experience, the more the developers tend to tailor design patterns.

TABLE VI. CORRELATION BETWEEN YEARS OF EXPERIENCE AND TAILORING DESIGN PATTERNS

	Category of Experience	Tailoring Category
Category of Experience	1	0.661712039
Tailoring Category	0.661712039	1

c) Recommendations to Improve Knowledge on Design Patterns

Fig. 2 illustrates the recommendations to improve knowledge on design patterns which are tabulated from the questionnaires. 45% of the respondents selected attending seminar or training as the most effective method. This shows that this is the best approach to better learn design patterns. However, online tutorials, websites, and books should not be ignored either as these are alternatives that can be used besides attending seminar or training.

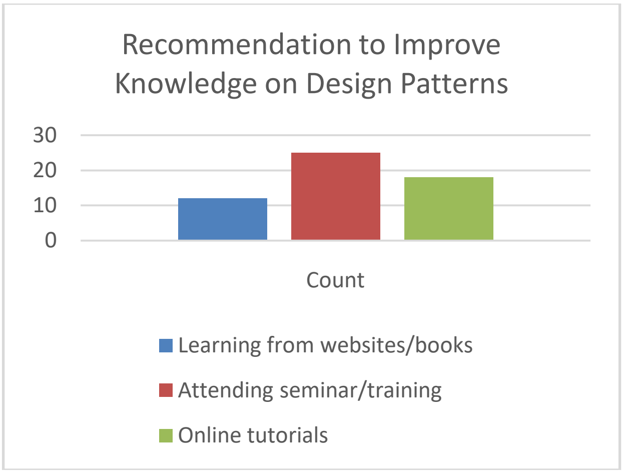

Fig. 2. Recommendations to Improve Knowledge on Design Patterns

Moreover, developers with higher years of experience mostly prefer to attend seminar or training compared to the other two. This further emphasized the strengths of seminar and training in helping developers to learn. This also serves as a good opportunity to exchange opinions with the experts. Therefore, the author would highly recommend for developers to go for seminar or training to deepen the knowledge on design patterns.

4.2 Proposed Framework

This section presents the proposed framework which is constructed based on the literature review and findings of the data analysis. The validity of this framework is evaluated using expert judgment and controlled experiment. The feedbacks collected from both evaluations are used to make refinements to further improve the framework. Fig. 3 displays the proposed framework which consists of three phases such as Problem Identification, Pattern Evaluation, and Solution Verification.

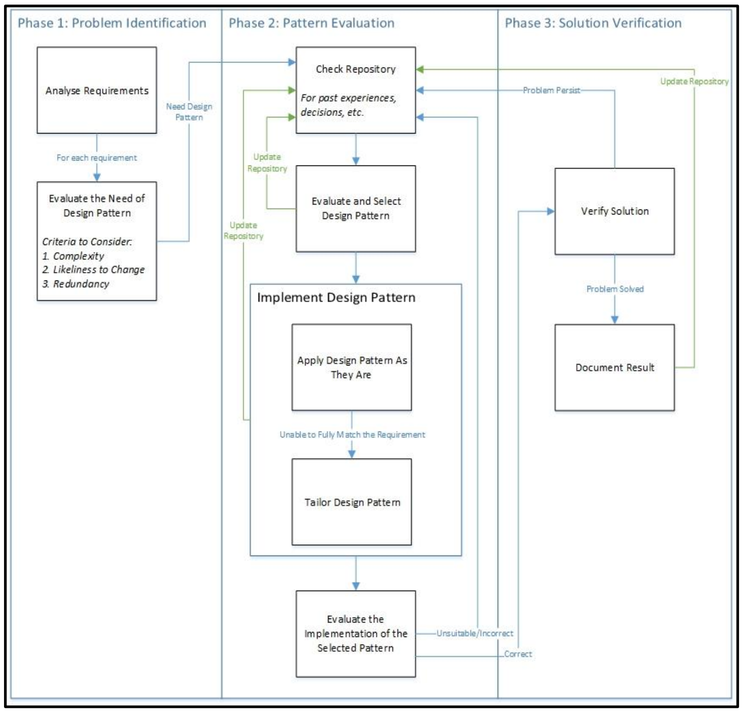

Fig. 3: Proposed Framework

a) Phase 1: Problem Identification

Analyse Requirements

Requirements are the main criteria to select a design pattern. Each requirement must be fully understood and critically evaluated. This is extremely important as it highly affects the next step where a decision whether to use design patterns will be made.

Evaluate the Need of Design Pattern

This step mainly checks if design pattern is needed to be applied to the requirement. Design pattern is less likely to be used for simple problem to avoid needlessly complex software design. If requirements are fixed and it is foreseen that it is very unlikely to have changes, then there is no need to apply design pattern. Furthermore, overusing design patterns will lead to antipatterns which result in poor system design. It makes the software more complex, thus increasing the difficulty to perform maintenance. Therefore, design patterns should only be used when necessary.

b) Phase 2: Pattern Evaluation

Check Repository

This step is to check for past experiences, decisions, etc. stored in a repository that would best fit the requirement. For instance, for a certain problem, a design pattern that is selected and tested to be not suitable is recorded in the repository. Then, developers can refer to existing data to avoid selecting the same pattern again for similar requirement. This will allow developers to make a more informed decision while selecting and implementing the design patterns. However, for first time use, this step can be skipped because there are no data being stored in the repository.

Evaluate and Select Design Pattern

Once it is confirmed that design pattern is needed to solve the problem, the next step is to evaluate the suitability of a pattern with the problem encountered before selecting it to avoid undo and redo. This is extremely important because system quality is reduced when inappropriate design pattern is used [30]. This must be done right the first time to avoid undo and redo as well as to obtain optimum result. Upon selecting a design pattern, all the related information will be updated to the repository.

Implement Design Pattern

Experienced developers tend to write codes that are more maintainable and understandable compared to beginners. Similarly, years of experience also has significant contribution on how design patterns are being implemented. Design patterns are often applied as they are and as experience increases, developers tend to tailor the patterns. However, this does not mean that developers with lower experience are not allowed to tailor design patterns. It simply shows that it is more common for higher experience developers to tailor patterns due to their familiarity and knowledge on design patterns.

In addition, tailoring design pattern should also be made when it is unable to match the requirement by applying the pattern as they are. All developers are encouraged to tailor design patterns to fit the requirement especially when applying the patterns as they are could not solve the issue. One of the most common tailoring is to combine various patterns into one to solve a certain problem. Tailoring design patterns basically is to customize the original patterns to match the requirement. It again falls back to the requirement as it is the main criteria to be focused on. Applying as they are or tailoring, it must be emphasized that design pattern must be implemented the right way. This is vital because wrong implementation will complicate software design [9]. Lastly, all the related information in this step (e.g. implementation and tailoring) are updated to the repository.

Evaluate the Implementation of the Selected Pattern

This evaluation makes this phase iterative, meaning that it requires developers to redo the activities from the beginning when it is done incorrectly. There are two conditions that determine if iteration is necessary such as inappropriate selection and wrong implementation of design pattern. Design patterns which are unsuitable will reduce system quality [30]. Software design will become more complex when the design pattern is implemented wrongly [9]. These two statements provide the basis on the need of this step. Code review is an example to conduct this evaluation which can be performed by individuals or group of developers together.

c) Phase 3: Solution Verification

Verify Solution

The purpose of this step is to ensure that the problem is fully solved using design pattern. Furthermore, testing the module or functionality developed is the key action to do so. Through testing, developers will be able to exploit hidden software bugs early and apply fixes before it causes more problems down the line. This not only reduces maintenance cost, but also helps to ensure the quality of the software developed. A high-quality software can then provide optimum customer satisfaction by bringing the best user experience. However, it is necessary to revisit Phase 2 when the problem is not solved. This is continuously repeated until the problem is fully solved.

Document Result

This step is to document the result of applying the selected pattern(s) on the problem faced into the repository. For example, how a particular problem is solved using the selected design pattern and what are the benefits gained from doing so. This information can be used for many purposes such as reporting, future references, etc.

V. CONCLUSION

Changes are inevitable in IT projects and how software is being designed matters a lot. Inflexible software design has caused serious software maintenance issue which lead to costly changes. Design patterns provide tested solutions to recurring problems. However, it is difficult to select the most suitable pattern and there is yet to be a proven guideline for this. The difficulty is faced by both beginners and experienced developers. The proposed framework is aimed to solve this problem and it is the main contribution of this dissertation. It consists of three phases with several activities which include analysis, evaluation, selection, implementation, and verification. The framework is constructed based on the literature review and data analysis. Then, it is validated and refined based on the collected feedbacks from expert judgments and controlled experiment. The main contribution of this framework is that it helps developers in evaluating and selecting the right pattern.

VI. LIMITATIONS

One of the limitations of the study is that the developed framework is more beneficial to users who already have knowledge on design patterns. It does not help developers on how to correctly implement design patterns. Developers are required to rely on their skills and experiences to implement it correctly. In addition, measurement with metrics are not included in this research as it is deemed too broad to be included because beginners and even experienced developers may not be able to fully utilize those metrics. Therefore, metrics will be added in the future work.

VII. Future Work

There are many ways to continue this research to researchers who interested in design patterns such as:

1) Adding variable(s) to the framework to further improve it.
2) Conduct research on design patterns other than the ones from GoF to validate the proposed framework.
3) Create guideline on how to correctly implement design patterns.
4) Include metrics for specific scenario to measure the result gained by referring to the framework.

VIII. References

[1] The Standish Group, "The Standish CHAOS Report 2014," *Proj. Smart*, p. 16, 2014.

[2] S. Dehaghani and N. Hajrahimi, "Which Factors Affect Software Projects Maintenance Cost More?," *Acta Inform. Medica*, vol. 21, no. 1, p. 63, 2013.

[3] Omnext BV, "How to save on software maintenance costs: An Omnext white paper on software quality," vol. 31, no. November, pp. 1–13, 2014.

[4] S. Holzner, *Design Patterns for Dummies®*. Indianapolis: Wiley Publishing, 2006.

[5] M. Oruc, F. Akal, and H. Sever, "Detecting design patterns in object-oriented design models by using a graph mining approach," *Proc. - 2016 4th Int. Conf. Softw. Eng. Res. Innov. CONISOFT 2016*, pp. 115–121, 2016.

[6] S. S. Thabasum and U. T. M. Sundar, "A Survey on Software Design Pattern Tools for Pattern Selection and Implementation," *Int. J. Comput. Sci. Commun. Networks*, vol. 2, no. 4, pp. 496–500, 2019.

[7] M. R. Jameel Qureshi and W. Al-Geshari, "Proposed Automated Framework to Select Suitable Design Pattern," *Int. J. Mod. Educ. Comput. Sci.*, vol. 9, no. 5, pp. 43–49, May 2017.

[8] R. Subburaj, J. Gladman, and C. Hwata, "Impact of Object Oriented Design Patterns on Software Development," *Int. J. Sci. Eng. Res.*, vol. 3, no. 2, pp. 961–967, 2015.

[9] M. O. Onarcan and Y. Fu, "A Case Study on Design Patterns and Software Defects in Open Source Software," *J. Softw. Eng. Appl.*, vol. 11, no. 05, pp. 249–273, 2018.

[10] C. Zhang and D. Budgen, "What do we know about the effectiveness of software design patterns?," *IEEE Trans. Softw. Eng.*, vol. 38, no. 5, pp. 1–19, 2012.

[11] C. Zhang, F. Wang, R. Xu, X. Li, and Y. Yang, "A quantitative analysis of survey data for software design patterns," in *Proceedings of the 2014 3rd International Workshop on Evidential Assessment of Software Technologies - EAST 2014*, 2014, no. 111, pp. 48–55.

[12] B. Walter and T. Alkhaeir, "The relationship between design patterns and code smells: An exploratory study," *Inf. Softw. Technol.*, vol. 74, pp. 127–142, Jun. 2016.

[13] F. M. Alghamdi and M. R. J. Qureshi, "Impact of Design Patterns on Software Maintainability," *Int. J. Intell. Syst. Appl.*, vol. 6, no. 10, pp. 41–46, Sep. 2014.

[14] K. Beck *et al.*, "Industrial Experience with Design Patterns," in *Proceedings of the 18th International Conference on Software Engineering*, 1996, pp. 103–114.

[15] M. Ali and M. O. Elish, "A Comparative Literature Survey of Design Patterns Impact on Software Quality," in *2013 International Conference on Information Science and Applications (ICISA)*, 2013, pp. 1–7.

[16] N. Ahmad and M. W. Boota, "Evaluation Amid different Software Design Patterns," *Int. J. Comput. Appl.*, vol. 105, no. 11, pp. 28–34, 2014.

[17] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-oriented Software*. Massachusetts: Addison-Wesley, 1995.

[18] D. Yu, P. Zhang, J. Yang, Z. Chen, C. Liu, and J. Chen, "Efficiently detecting structural design pattern instances based on ordered sequences," *J. Syst. Softw.*, vol. 142, pp. 35–56, 2018.

[19] V. Holmstedt and S. A. Mengiste, "The importance of program Design Patterns training," in *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2017, pp. 559–560.

[20] W. B. McNatt and J. M. Bieman, "Coupling of design patterns: common practices and their benefits," in *25th Annual International Computer Software and Applications Conference. COMPSAC 2001*, 2001, no. Compsac, pp. 574–579.

[21] P. Wendorff, "Assessment of design patterns during software reengineering: lessons learned from a large commercial project," in *Proceedings Fifth European Conference on Software Maintenance and Reengineering*, 2001, pp. 77–84.

[22] F. Khomh and Y.-G. Gueheneuce, "Do Design Patterns Impact Software Quality Positively?," in *2008 12th European Conference on Software Maintenance and Reengineering*, 2008, pp. 274–278.

[23] P. Hegedüs, D. Bán, R. Ferenc, and T. Gyimóthy, "Myth or Reality? Analyzing the Effect of Design Patterns on Software Maintainability," in *Communications in Computer and Information Science*, 2012, vol. 340, pp. 138–145.

[24] F. Khomh and Y.-G. Gueheneuc, "Design Patterns Impact on Software Quality : Where Are the Theories ?," in *IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2018, pp. 15–25.

[25] H. Marouane, C. Duvallet, A. Makni, R. Bouaziz, and B. Sadeg, "An UML profile for representing real-time design patterns," *J. King Saud Univ. - Comput. Inf. Sci.*, vol. 30, no. 4, pp. 478–497, 2017.

[26] A. Abdullah, M. H. Khan, and R. Srivastava, "Flexibility: A Key Factor to Testability," *Int. J. Softw. Eng. Appl.*, vol. 6, no. 1, pp. 89–99, Jan. 2015.

[27] C. Gravino and M. Risi, "How the Use of Design Patterns Affects the Quality of Software Systems: A Preliminary Investigation," in *2017 43rd Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, 2017, pp. 274–277.

[28] G. Scanniello, C. Gravino, M. Risi, G. Tortora, and G. Dodero, "Documenting Design-Pattern Instances: a Family of Experiments on Source Code Comprehensibility," *ACM Trans. Softw. Eng. Methodol.*, vol. 24, no. 3, pp. 1–35, May 2015.

[29] M. N. Riaz, "Impact of software design patterns on the quality of software: A comparative study," in *2018 International Conference on Computing, Mathematics and Engineering Technologies (iCoMET)*, 2018, pp. 1–6.

[30] H. Zhu and I. Bayley, "On the Composability of Design Patterns," *IEEE Trans. Softw. Eng.*, vol. 41, no. 11, pp. 1138–1152, Nov. 2015.