Jurnal
Teknologi

Full Paper

EVALUATION OF THE VISITOR PATTERN TO PROMOTE SOFTWARE DESIGN SIMPLICITY

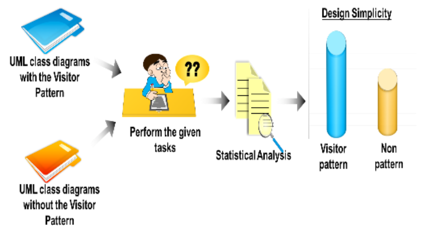
Aziz Nanthaamornphong*, Rattana Wetprasit

Department of Information and Communication Technology, Faculty of Technology and Environment, Prince of Songkla University, Phuket Campus, Thailand

Article history
Received
2 February 2015
Received in revised form
14 September 2015
Accepted
12 October 2015

*Corresponding author
aziz.n@phuket.psu.ac.th

Graphical abstract


UML class diagrams with the Visitor Pattern
UML class diagrams without the Visitor Pattern
??
Perform the given tasks
Statistical Analysis
Design Simplicity
Visitor pattern
Non pattern

Abstract

Design patterns, which have been widely used by software engineering communities, have been claimed to improve software design in previous studies. However, there is little empirical evidence to support such a claim. Additionally, the benefits of design patterns in software design have not been studied in sufficient detail to date. As a result, in this study, we used empirical methods to evaluate whether design patterns help developers improve the simplicity of software design. In particular, we analyzed how easily a given software design was understood. We chose the well-known Visitor pattern as the design pattern for this study. The results suggest that the Visitor pattern could help developers improve software design simplicity. Specifically, a class diagram with the Visitor pattern was found to be easier to understand than a class diagram without the design pattern.

Keywords: Design patterns, empirical study, software quality

Abstrak

Corak reka bentuk, yang telah digunakan secara meluas oleh komuniti kejuruteraan perisian, telah dituntut sebagai mampu meningkatkan reka bentuk perisian dalam kajian sebelum ini. Walau bagaimanapun, hanya terdapat sedikit bukti empirikal untuk menyokong tuntutan tersebut. Selain itu, manfaat corak reka bentuk dalam reka bentuk perisian belum dikaji secara terperinci setakat ini. Oleh itu, dalam kajian ini, kami menggunakan kaedah empirikal untuk menilai sama ada corak reka bentuk membantu pembangun sistem meningkatkan kesederhanaan dalam reka bentuk perisian. Khususnya, kami telah menganalisa betapa mudahnya reka bentuk perisian yang diberikan untuk difahami. Kami memilih corak Visitor yang terkenal sebagai corak reka bentuk untuk kajian ini. Keputusan menunjukkan bahawa corak Visitor boleh membantu pembangun sistem meningkatkan kesederhanaan dalam reka bentuk perisian. Secara khususnya, gambarajah kelas dengan corak Visitor didapati lebih mudah untuk difahami daripada gambar rajah kelas tanpa corak reka bentuk.

Kata kunci: Corak reka bentuk, kajian empirikal, kualiti perisian

1.0 INTRODUCTION

In the past several decades, many software engineering studies have reported that software design patterns improved software quality [1], particularly for object-oriented software [2]. Researchers have studied various aspects of design patterns, such as their characteristics, applicability, benefits and drawbacks [3-6].

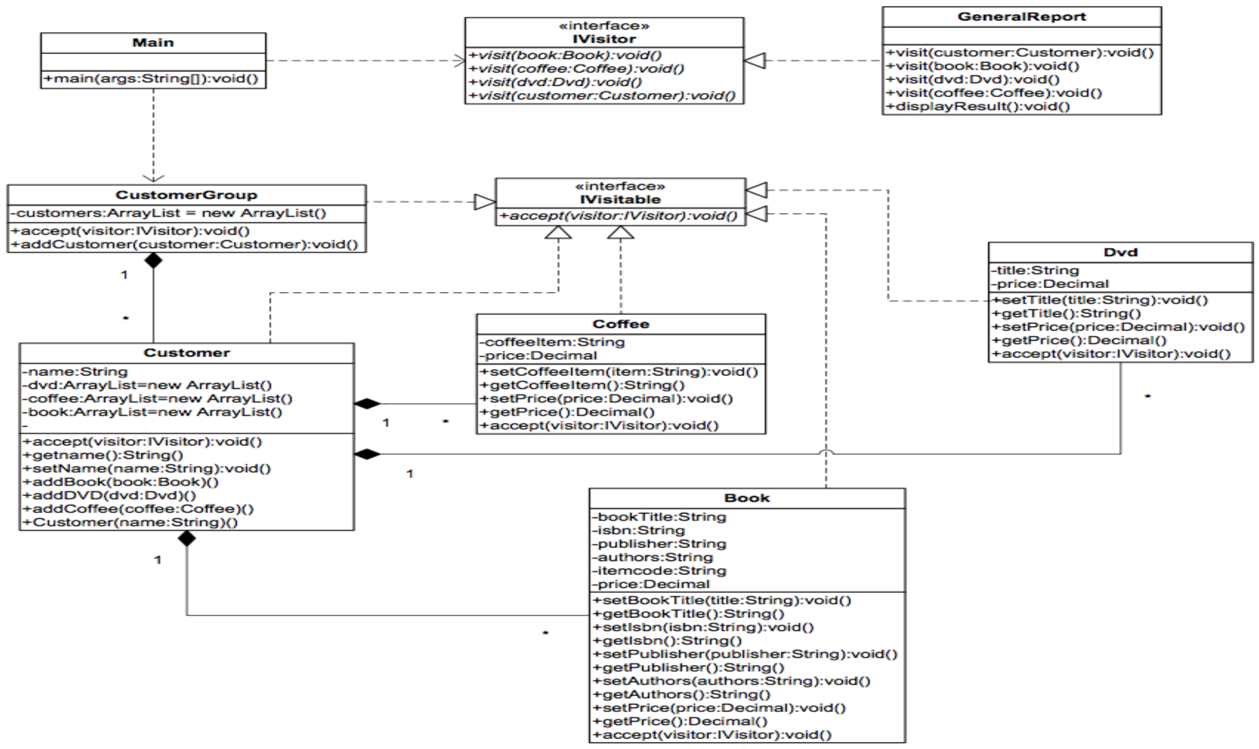

Main
+main(args:String[]):void()
«interface»
IVisitor
+visit(book:Book):void()
+visit(coffee:Coffee):void()
+visit(dvd:Dvd):void()
+visit(customer:Customer):void()
GeneralReport
+visit(customer:Customer):void()
+visit(book:Book):void()
+visit(dvd:Dvd):void()
+visit(coffee:Coffee):void()
+displayResult():void()
CustomerGroup
-customers:ArrayList = new ArrayList()
+accept(visitor:IVisitor):void()
+addCustomer(customer:Customer):void()
«interface»
IVisitable
+accept(visitor:IVisitor):void()
Dvd
-title:String
-price:Decimal
+setTitle(title:String):void()
+getTitle():String()
+setPrice(price:Decimal):void()
+getPrice():Decimal()
+accept(visitor:IVisitor):void()
Coffee
-coffeeItem:String
-price:Decimal
+setCoffeeItem(item:String):void()
+getCoffeeItem():String()
+setPrice(price:Decimal):void()
+getPrice():Decimal()
+accept(visitor:IVisitor):void()
Customer
-name:String
-dvd:ArrayList=new ArrayList()
-coffee:ArrayList=new ArrayList()
-book:ArrayList=new ArrayList()
+accept(visitor:IVisitor):void()
+getname():String()
+setName(name:String):void()
+addBook(book:Book)()
+addDVD(dvd:Dvd)()
+addCoffee(coffee:Coffee)()
+Customer(name:String)()
Book
-bookTitle:String
-isbn:String
-publisher:String
-authors:String
-itemcode:String
-price:Decimal
+setBookTitle(title:String):void()
+getBookTitle():String()
+setIsbn(isbn:String):void()
+getIsbn():String()
+setPublisher(publisher:String):void()
+getPublisher():String()
+setAuthors(authors:String):void()
+getAuthors():String()
+setPrice(price:Decimal):void()
+getPrice():Decimal()
+accept(visitor:IVisitor):void()
1
*
1
*
1
*
1
*

Figure 1 Example of the design diagram

In previous studies, design patterns were not always found to improve software quality. For example, a complex implementation of design patterns can negatively impact software quality (e.g., software maintenance, program comprehension). In addition, although design patterns allow easy modification of software, they are often expensive [7, 8].

Most studies also primarily considered the implementation and maintenance phases of software projects. Meanwhile, only a few studies have focused on the design or early activities in the software development process [9]. Additionally, only a small amount of empirical evidence has been reported to support the claim of improved simplicity [10] when design patterns are used in software design.

Therefore, this study examines whether Visitor patterns help developers increase the simplicity of their software designs. Simplicity is an important quality attribute of a software system, particularly object-oriented software systems. We can define the simplicity of a software design as the degree to which the design of a software system can be easily understood [11]. With the Visitor design pattern, a visitor "represents an operation to be executed on the elements of an object structure. The Visitor allows the developers to make a new operation without altering the classes of the elements on which it operates" [12]. Over the years, the Visitor pattern has been the topic of several studies [13] but few empirical studies were conducted to support it. We chose to study the Visitor pattern because this pattern is broadly used in practice and has several design choices.

We studied how the Visitor pattern affected the simplicity of software design. We asked subjects to first look at two types of designs, one with the Visitor pattern and the other without it, and then complete questionnaires related to the designs. The designs used were UML class diagrams, which are a de facto standard, and their primary constructs are well known in the software engineering community. We developed design diagrams for two different systems, which one of those diagrams is shown in Figure 1. We used the task duration and correctness to compare the simplicity of the designs and thereby determine quantitatively whether the Visitor pattern could improve the design simplicity. Additionally, we conducted a survey, an empirical strategy used in software engineering research [14], after each experiment to increase our confidence in the analyzed results.

The remainder of this article is organized as follows. Section 2 provides an overview of related works. Section 3 describes the experimental methods used in this study. Section 4 explains the post-study, in which the main experiment was extended. The data analysis and its interpretation are discussed in Section 5. Section 6 outlines the validation and limitations of this study. Finally, conclusions and future work are described in Section 7.

2.0 RELATED WORKS

This study is related to previous studies on how design patterns affect software quality.

The relationship between faults and design patterns has been investigated by Mahmoud Elish and Mawal Mohammed [15], who examined the faults in five open-source software systems. The researchers compared the faults between classes that participated in the design patterns and those that did not. The results showed an inconsistent difference in the number of faults between the participant and non-participant classes in the design patterns.

Sfetsos *et al*. performed an empirical study to investigate how 23 design patterns used in software libraries and standalone applications improved software quality [16]. In their study, software quality was measured using QMOOD metrics. The results of that study indicated that the Visitor patterns did not have a positive effect on any software quality attributes.

Sebastien J. *et al*. conducted an empirical study to investigate the influence of the Visitor pattern on program comprehension and maintenance [17]. The study showed that the Visitor pattern only improved modification tasks. This study is difficult to replicate because the researchers used an eye-tracker to measure cognitive loads while the subjects were performing the tasks. Additionally, we believe that the use of this eye-tracker might affect how the subjects performed on the given tasks.

Kmomh and Gueheneuc also studied the influence of 23 design patterns on ten different quality attributes. Their study indicated that the design patterns did not always improve software quality [18]. More specifically, the results showed that patterns did not conclusively promote reusability, expandability, and understandability. This study highlighted the need to assess the effects of a design pattern on other software quality attributes.

Javier Garzas *et al*. performed a controlled experiment to investigate whether design patterns could help developers easily understand and easily modify software designs [11]. The results suggest that more effort is required to change the design for a diagram including design patterns than for a diagram without design patterns.

Wendorff evaluated the advantages of design patterns applied to large commercial software [9] but did not study the effect of design patterns on software quality. This study indicated that design patterns did not necessarily enhance software designs. Additionally, the results showed that a design pattern could increase complexity and that removing patterns was expensive. However, this study only provided qualitative data.

Although several other empirical studies have investigated the effect of design patterns on software quality, their conclusions are not consistent. There is also lack of additional empirical studies analyzing the influence of design patterns on software design. Rather than focusing on the source-code level, as done in other studies, this study focused on the design level.

3.0 EXPERIMENTAL SETTING

In this section, we explain the experiment used to evaluate the influence of the Visitor pattern on the simplicity of software design.

3.1 Hypotheses

The primary goal of this experiment was to compare the simplicity of software designs with and without the Visitor pattern. The primary question of this study can be expressed as follows:

Are software designs using the Visitor pattern easier to understand than software designs without a design pattern?

We established the hypotheses for the proposed experiment as follows:

- H_{0Time}: Using the Visitor pattern in the design does not improve the time required to understand the design.
- H_{1Time}: Using the Visitor pattern in the design improves the time required to understand the design.
- H_{0Easy}: Using the Visitor pattern in the design does not make it easier to understand the design.
- $H1Easy$: Using the Visitor pattern in the design makes it easier to understand the design.
- H_{0Eff}: Using the Visitor pattern in the design does not improve the understandability efficiency.
- H_{1Eff}: Using the Visitor pattern in the design improves the understandability efficiency.

We used a one-sided alternative hypothesis because the Visitor pattern can be considered worthwhile if it can produce better results than the non-design pattern version. In other words, the Visitor pattern should only be used if it can improve the software design. The empirical evidence obtained in this study should prove or refute the hypotheses detailed above.

3.2 Independent Variables

Based on the hypotheses detailed in Section 3.1, we identify the following independent variable:

- *Software Design*. There were four different designs on two different systems, two of which used a Visitor pattern (VP) and two of which did not (NP).

We developed four design diagrams, which were presented via UML class diagrams. These class diagrams were built on two different software systems: a reporting module in an entertainment store and a patient behavior information system in a hospital. The details of these designs and their class diagrams are shown in Appendix A. We named the entertainment store system and the hospital system "System A" and "System B" respectively. Note that Figure 1 and Figure 9 show the VP and NP versions of System A, respectively, and Figure 10 and Figure 11 show the VP and NP versions of System B. We decided to develop these two system designs ourselves because other available systems, such as JHotDraw and Eclipse, would be too complex for the subjects participating in this study. Additionally, we wanted to ensure that the selected systems only use the Visitor pattern.

3.3 Dependent Variables

The dependent variables of this experiment included the following:

- *Time*. We measured the time (minutes) the subjects spent answering the six questions related to the software design.
- *Correctness*. This variable was the score assigned to quantify the correctness of the subject's answers. We established a specific score for each question. All questions were open-ended, so the grading was determined based on specific keywords expected in the answers. Answers containing these keywords received the assigned score, and those without these keywords received a score of zero. The questions were carefully constructed by two authors so that the scoring was straightforward and unambiguous.
- *Efficiency*. Simplicity was measured by how easily the designs were understood. A design was considered easier to understand if the subjects could obtain a high score in a shorter period of time. The efficiency was obtained by dividing each subject's score by the time spent answering the question.

We measured the design simplicity as the correctness and time required because we believed that the subjects would only be able answer a question correctly and quickly if the design being evaluated were easy to understand. Although some subjects might answer correctly by guessing, this was fairly unlikely [19]. We chose the efficiency as another important measure because it is the measure most frequently used by software engineers in the software development process. Appendix B shows the details of the questionnaire used in this study. Two questionnaires (Questionnaire A and B) were used, where Questionnaire A asked the subjects about diagrams 1 and 2, and Questionnaire B asked questions about diagrams 3 and 4. In each questionnaire, each subject had to note the time at which he/she began to perform the given task and the time at which he/she finished the task.

3.4 Subjects

A total of 24 subjects participated in the experiment. These subjects were undergraduate students enrolled in a software engineering class taught by one of the authors.

3.5 Experimental Procedure

In this experiment, each subject completed tasks for both treatments: VP and NP. This experiment was designed such that each subject performed the tasks on both design versions according to a repeated-measures design. The dependent variables of each subject's performance were repeatedly measured after the subject completed the task for each software design version in the experiment. Exposing each subject to both treatments allowed us to reduce inter-subject differences, which can be appear as random error in the subject scores.

We assigned the 24 subjects into four balanced groups, as shown in Figure 2. To minimize the order effect, which describes how the order of performing

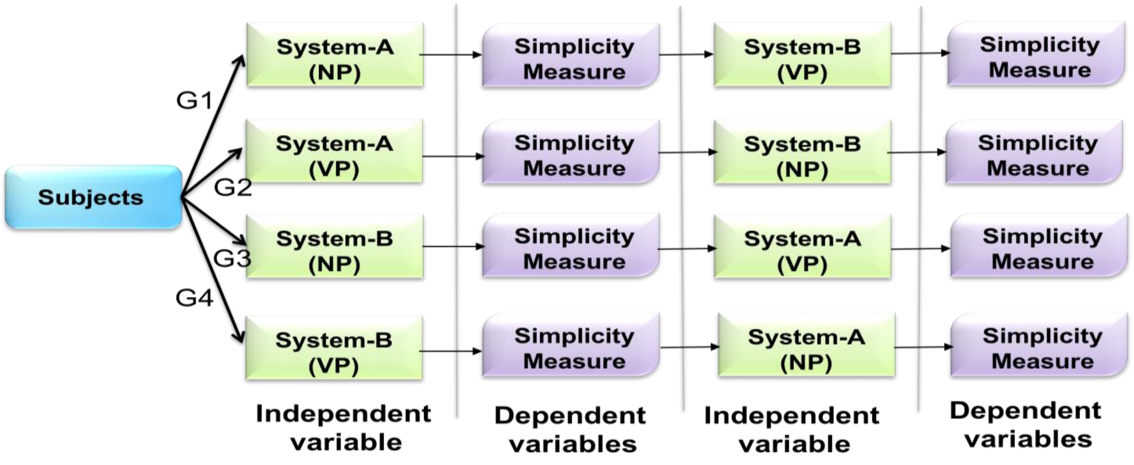


Figure 2 Experiemental setting

a task affects the dependent variable, each subject had to answer the questions on two different class diagrams designed for different systems. For example, the subjects in G1 answered the questions based on the NP version for System A and the VP version for System B.

Table 1 Questionnaire distribution

Group No.	No. of Subjects	Q-A	Q-B
1	6	Diagram#2	Diagram#3
2	6	Diagram#1	Diagram#4
3	6	Diagram#1	Diagram#3
4	6	Diagram#2	Diagram#4

Each subject performed the given tasks by answering six questions for the NP design and six questions for the VP design. All subjects answered the same questions. Each of the four groups answered the questions from Questionnaire A (Q-A) and Questionnaire B (Q-B) based on the diagram numbers, as shown in Table 1. Questionnaire A was designed to ask questions related to System A, and Questionnaire B was designed to ask questions related to System B. We developed each class diagram such that it contained sufficient information to answer the questions. The subjects were asked specific questions related to classes that appeared in the class diagram.

During the experiment, we allowed the subjects to refer to their textbooks or notes. The first author also instructed the subjects to notify the researcher if they had any trouble understanding the questions. We organized the room as though the subjects were taking an exam; thus, the subjects were separated spatially by empty seats, preventing them from talking to each other. Additionally, the subjects could not leave the room without permission.

We ran a pilot study before the complete experiment to validate and verify the proposed experimental method. We selected six subjects who were not involved in the real experiment to participate in the pilot study. We asked the subjects to follow the experimental procedure detailed above. The pilot study allowed us to determine how well the subjects performed the given tasks. We then evaluated the preliminary results of the pilot study in terms of whether the questionnaires were understood and whether the answers were sufficient for analysis. We then revised the questionnaire based on the results of the pilot study.

4.0 POST-STUDY

To better understand other factors that might affect the experimental results; we conducted a survey of the subjects two weeks after they finished the given tasks. This section describes this additional study.

We sent each subject a survey that included questions regarding the following aspects of the primary experiment:

1) Knowledge of the UML class diagram. To ensure that all subjects were familiar with UML class diagram notation (e.g., box, line), we asked the subjects whether they had ever used a UML class diagram.

2) Ease of reading each diagram. These questions were posed to investigate whether the layout of given diagrams impacted the subject's task performance.

3) Experience with the Visitor pattern before the experiment. We believe that subjects with experience with the pattern would perform differently than those without such experience.

4) Confidence in their answers. These questions were posed to ensure that the subjects provided answers without guessing.

Table 2 Descriptive statistics and t-Test results

Variables	Visitor Pattern		Non-Pattern		t	Degree of Freedom	Sig. (p-value)	Effect Size
	Mean	Standard Deviation	Mean	Standard Deviation				
Time (min)	11.43	1.08	12.23	0.80	2.391	23	0.025	0.82
Correctness	3.96	1.16	2.63	1.44	2.844	23	0.009	1.02
Efficiency	0.35	0.12	0.22	0.13	-2.939	23	0.007	1.04

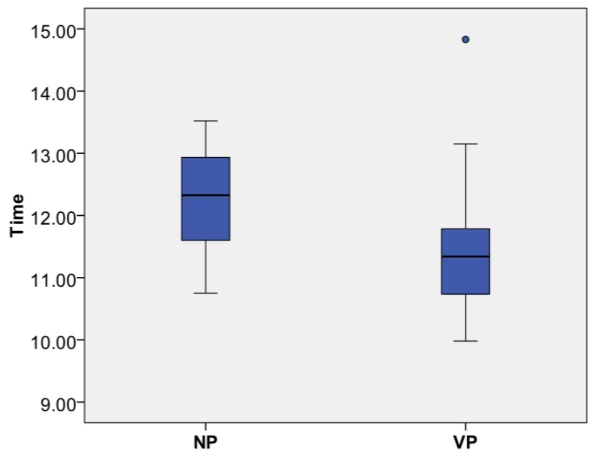


Figure 3 Distribution of spent time in minutes

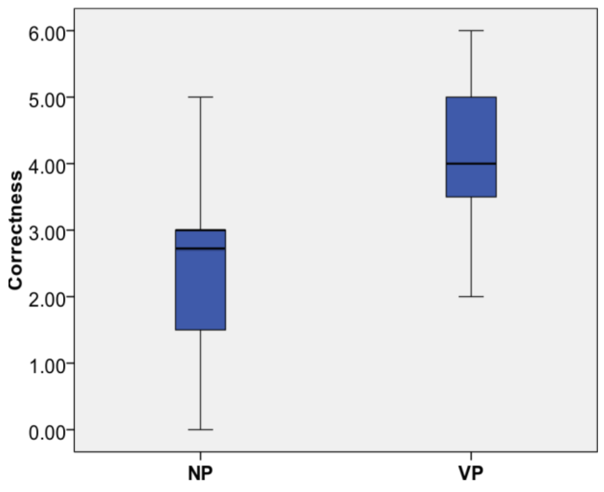


Figure 4 Distribution of correctness

We used a self-developed questionnaire technique that included two different formats: dichotomous Yes/No answers and self-assessment items that used a five-point scale. Appendix C presents the survey questions. We distributed the survey via email along with the given tasks, including the questions and diagrams. To ensure that the survey questions were comprehensible and valid with respect to the study construct, we asked experts to evaluate the proposed survey questions. We then refined the questions based on feedback. After the verification process, we made the survey available on the web, where it could be accessed via a URL. Finally, we distributed the URL via email. We also posted a reminder in the News section of the department website.

5.0 RESULTS AND DISCUSSION

This section presents the results based on data obtained from the experiment. The measurements of the ease of understanding the design using the Visitor pattern were the correctness, time spent, and the

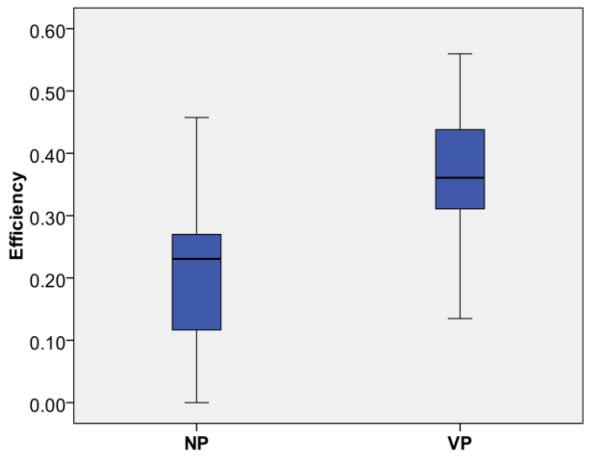


Figure 5 Distribution of efficiency

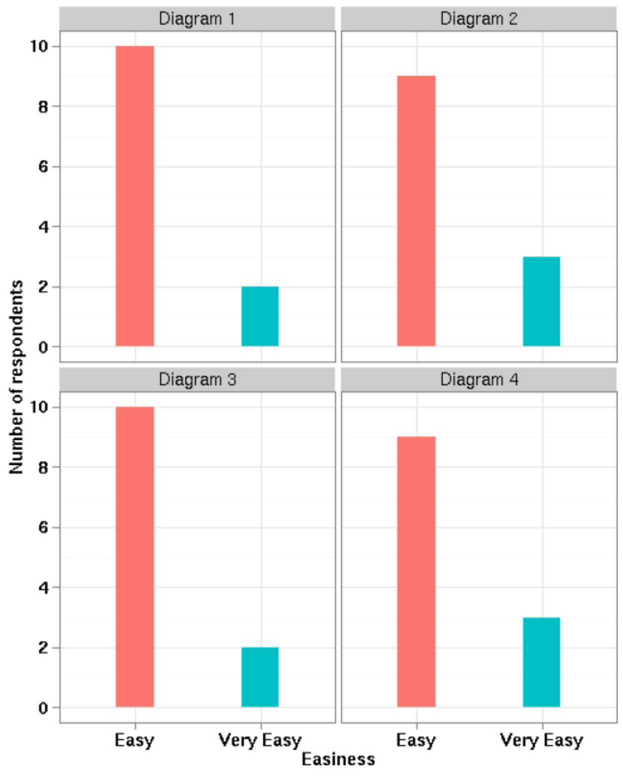


Figure 6 The easiness of the diagram's layout

efficiency of the subjects. Regarding the correctness, the subjects received one point for a correct answer and zero points for an incorrect answer. The task time is the time that the subject required to complete the questions. Thus, the efficiency is the number of correct answers divided by the task time.

After the experiment was complete, we received the questionnaires from all 24 subjects. Each subject responded to all of the questions in both questionnaires. Therefore, the quantity of data collected was sufficient for analysis. Once we obtained the results, we performed a descriptive analysis of the data. Table 2 presents the main descriptive statistics for the experiment.

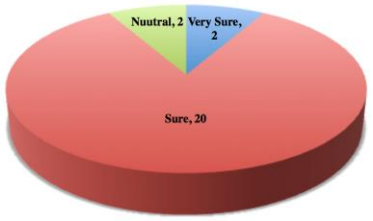


Figure 7 The confidence of Questionnaire-A

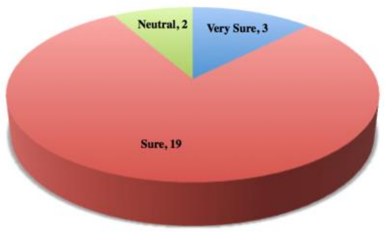


Figure 8 The confidence of Questionnaire-B

The results of the experiment showed that less time was spent on the VP questionnaire than the NP questionnaire. When using the Visitor pattern, the subjects also answered the questions 50.5% more accurately (means: 3.95 for VP and 2.63 for NP, corresponding to a positive difference of 1.33), and the efficiency was improved by 60.3% (means: 0.35 for VP and 0.22 for NP, corresponding to a positive difference of 0.13).

Figure 3 shows the distribution of the task time; the median for VP was approximately 11.5 minutes, which is lower than the median for NP. This result is in agreement with the correctness results, shown Figure 4, in which the median for VP was approximately 4. Similarly, the median efficiency for VP, shown in Figure 5, is 0.35. These values suggest that the design with the Visitor pattern was easier to understand.

Next, we examined whether these differences were statistically significant. We analyzed the hypotheses tests using a t-test with a significance level of 0.05. The last four columns in Table 2 show the statistical results for the hypothesis testing of the dependent variables. Note that we also measured the effect size of the statistical analysis with using the Cohen's d method, which is often used in controlled experiments [20].

H_{0Time} - When the Visitor pattern was used in the software design, a significantly shorter task time (p-value = 0.025, < 0.05) was observed. Therefore, we concluded that the null hypothesis H_{0Time} could be rejected. Regarding the effect size, the calculated d value (d=0.82, large) indicated a high practical significance. Thus, we concluded that the Visitor pattern reduced the time spent answering the questions on the software designs.

H_{0Easy} - The difference in the number of the correct answers was statistically significant (p-value = 0.009 < 0.05). Thus, we could reject the null hypothesis H_{0Easy}. The effect size (d = 1.02, large) also suggested a high practical significance. Consequently, we concluded that the software design using the Visitor pattern was easier to understand than the design without any pattern.

H_{0Eff} - The difference in efficiency was also statistically significant (p-value = 0.007 < 0.05). Therefore, the null hypothesis H_{0Eff} could be rejected. The effect size (d=1.04, large) also indicated a high practical significance. Therefore, the design diagram with the Visitor pattern improved the understandability efficiency.

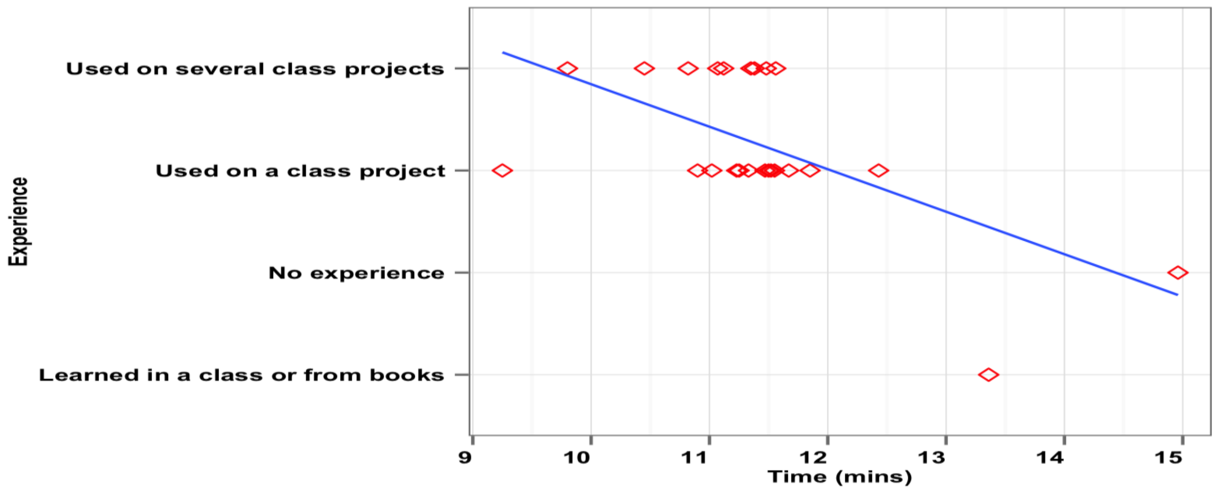


Figure 9 Correlation between time spent and experience

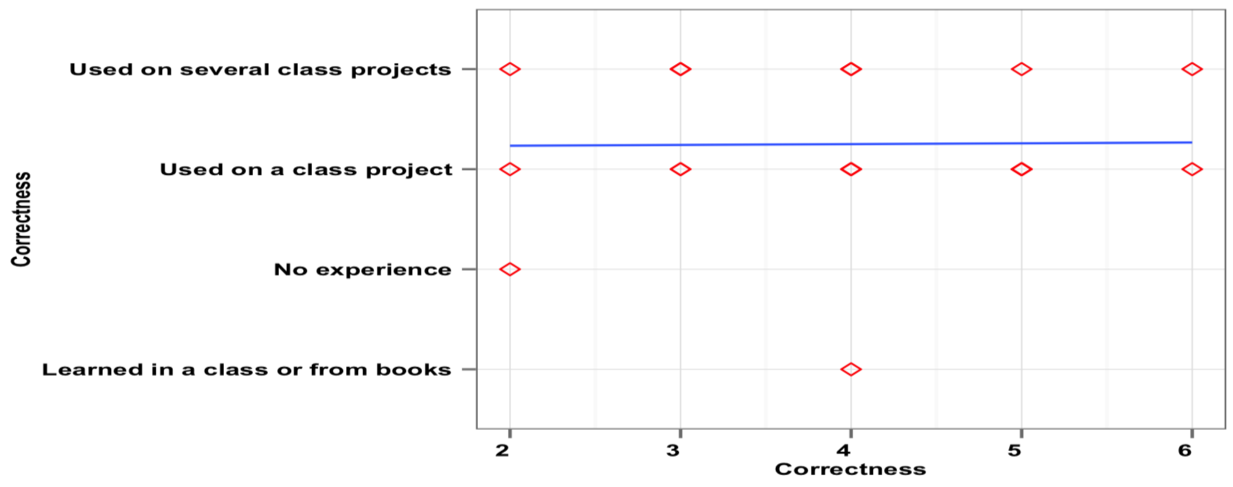


Figure 10 Correlation between correctness and experience

In the post-study, we received survey answers from all 24 subjects. The survey shows that subjects had used the UML class diagram prior to the experiment. Thus, the notation in the diagrams should not have influenced the subjects' performance during the experiment. Furthermore, we analyzed the effect of the diagram layout and the pattern presentation. All the subjects responded that the layout of each diagram was either easy or very easy to understand (Figure 6). These results conform to the two following questions, which asked the subjects to rate their confidence in their answers given during the experiment. Figure 7 and 8 summarize the responses to these questions. Overall, the subjects were confident in their answers; therefore, we exclude these questions from the proposed analysis.

For the statistical analysis, we used the Spearman rank-order correlation, which is a non-parametric statistic, to measure the strength and direction of the association between the variables, including 1) the task time and the subject's experience and 2) the correctness and the subject's experience. We only analyzed the results for the Visitor pattern group because we believed that the subject's experience should influence the tasks based on the design patterns.

Figure 9 shows the correlation between the time spent on the Visitor pattern diagrams and the subject's experience. Based on the statistical analysis, we found a strong, significant correlation ($rs = -0.48$, $p = 0.018$) between the time spent and the subject's experience. This evidence implies that as experience increases, the time spent on the given tasks decreases. Conversely, the subjects who had little experience might spend much more time on the given tasks.

However, no significant correlation was found between the correctness and experience level ($rs = 0.49$, $p = 0.822$), which is shown in Figure 10. Thus, experience likely does not affect the correctness. Note that Figure 9 and 10 do not include the answer "Used in a real industry project" because none of the subjects gave this response.

6.0 THREATS TO VALIDITY

Construct Validity. The construct validity is a generalization of the experimental results to the concept behind the experiment. The ease of understanding the design was measured by the task time, correctness, and efficiency, which are commonly used in empirical and comprehension studies in software engineering [8]. The data collected from the questionnaires might be inaccurate due to errors made by the subjects; for example, the subjects might not have noted the real start and end times in the questionnaires despite being instructed on how to complete the questionnaires. Additionally, in this experiment, we did not ask experts to evaluate the questionnaires; thus, it is possible that some questions were ambiguous, which could influence the subjects' answers. However, the information obtained from the pilot study was used to minimize this problem. Additionally, in the open-ended questionnaires, the researchers might have introduced a bias in favor of certain responses.

Internal Validity. Internal validity refers to the presence of other factors that might have an effect on the variables. Three common problems may have arisen during the experiment. First, the subjects might not have followed the prescribed process and instructions. Second, the subjects might not have been motivated to complete the questionnaires. However, we believe that these two problems were not serious because the questionnaires were part of an in-class assignment. Third, the subjects might have become fatigued because of the number of tasks that they were required to complete. However, we also do not believe that this fatigue effect was a serious problem because the maximum time allotted to complete the tasks was 30 minutes. Regarding the subject's knowledge of software domains, the designs were from different domains, but the researchers, as instructors of this course, knew that the subjects were generally familiar with these designs. Thus, knowledge of the domain did not affect the internal validity of this study. However, we did not completely address certain threats to the validity of this experiment. First, we did not assess the experience of the subjects on the Visitor pattern and their knowledge of UML prior to the experiment. Second, we did not carefully consider the class diagram layout, which might affect the ease of understanding the design represented in the diagrams. We mitigated these threats by conducting a survey inquiring about those concerns after the experiment. Although the subjects answered the survey after they finished the experiment, the answers should still represent their real knowledge as accurately as a survey conducted before the experiment would have. Thus, the results of the post-survey are likely similar to those of a pre-survey or pre-examination.

External Validity. To assess this type of validity, we considered the generalizability of the proposed study. Because the subjects in this experiment were students, the results might not be applicable to the software industry because students likely have less experience than professionals. Ideally, professionals would obtain equal or higher scores in shorter time periods than students would. Comparing the proposed design diagrams to industrial design diagrams, the proposed design diagrams are somewhat smaller and simpler. An additional study with professionals on real software designs might help us mitigate these threats. Additionally, we only used a t-test analysis to analyze the results, but the power of this significance test may be inconclusive.

7.0 CONCLUSION

The primary goal of this study was to evaluate whether the Visitor pattern promotes design simplicity. We quantified simplicity using measures of correctness, time and efficiency. We hypothesized that the Visitor pattern would have an effect on the software design only if these measures were significantly changed.

The experiment showed that a design with the Visitor pattern was easier to understand than a design without it. The differences between the use and non-use of the Visitor pattern were statistically significant with regard to correctness, time, and efficiency.

We also found that the Visitor pattern improved the design simplicity. Designers who decide to use the Visitor pattern in a specific context may consider using the Visitor pattern in future designs. Other experiments are required to verify the results of this study and extend these results to other design patterns.

In the future, we will replicate this empirical study to increase the external validity (i.e., using professionals instead of students, asking different questions, analyzing the complexity and layout of the design diagrams). We would also like to study the influence of the combination of design patterns. The primary goal of this future study will be to investigate the interactions of two or more design patterns in the same software design, as a real software design might contain many design patterns. For example, we will combine the Visitor pattern with other design patterns, such as the Abstract Factory pattern, the Singleton pattern or the Decorator pattern. Such combinations might reveal the effect of the interaction produced by each design pattern on the ease of understanding a given design. We also look forward to measuring the ease of understanding a design pattern via software metrics, which relates to the ease of understanding a given software design.

Finally, we believe that good software designs will directly help software engineers improve software quality. We also plan to study the effect of design patterns on other software quality attributes, such as modifiability, testability, and security.

Acknowledgement

The authors gratefully thank all participants in the experiment. All feedbacks from the 8th Malaysian Software Engineering Conference (MySEC) were extremely valuable.

References

[1] Venners, B. 2005. How to Use Design Patterns A Conversation with Erich Gamma, Part I. [Online]. From: http://www.artima.com/lejava/articles/gammadp.html. [Accessed on 7 September 2015].

[2] Beck, K., R. Crocker, G. Meszaros, J. Vlissides, J. O. Coplien, L. Dominick, and F. Paulisch. 1996. Industrial Experience with Design Patterns. *The 18th International Conference on Software Engineering*. Berlin, Germany. 25-30 March. 103-114.

[3] Prechelt, L., B. Unger-Lamprecht, M. Philippsen, and W. F. Tichy. 2002. Two Controlled Experiments Assessing the Usefulness of Design Pattern Documentation in Program Maintenance. *IEEE Transactions on Software Engineering*. 28(6): 595-606.

[4] Aversano, L., G. Canfora, L. Cerulo, C. Del Grosso, and M. D. Penta. 2007. An Empirical Study on the Evolution of Design Patterns. *The 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering*. Cavat near Dubrovnik, Croatia. 3-7 September 2013. 385-394.

[5] Mak, J. K. H., C. S. T. Choy, and D. P. K. Lun. 2004. Precise Modeling of Design Patterns in UML. *The 26th International Conference on Software Engineering*. Scotland, UK. 23-28 May 2004. 252-261.

[6] Ng, T. H., S. C. Cheung, W. K. Chan, and Y. T. Yu. 2006. Work Experience versus Refactoring to Design Patterns: A Controlled Experiment. *The 14th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. Oregon, USA. 5-11 November 2006. 12-22.

[7] Prechelt, L., B. Unger, W. F. Tichy, P. Bro ̈ssler, and L. G. Votta. 2001. A Controlled Experiment in Maintenance Comparing Design Patterns to Simpler solutions. *IEEE Transactions on Software Engineering*. 27(12): 1134-114.

[8] McNatt, W. and J. Bieman. 2001. Coupling of Design Patterns: Common Practices and Their Benefit. *The 25th*

Annual International Conference on Computer Software and Application. Illinois, USA. 8-12 October 2001. 574-579.

[9] Wendorff, P. 2001. Assessment of Design Patterns During Software Reengineering: Lessons Learned from a Large Commercial Project. *The 5th European Conference on Software Maintenance and Reengineering*. Libson, Portugal. 14-16 March 2001. 77-84.

[10] Zhang C. and D. Budgen. 2012. What Do We Know about the Effectiveness of Software Design Patterns? *IEEE Transactions on Software Engineering*. 38(5): 1213-1231.

[11] Garza´s, J., F. Garćıa, and M. Piattini. 2009. Do Rules and Patterns Affect Design Maintainability? *Journal of Computer Science and Technology*. 24(2): 262-272.

[12] Gamma, E., R. Helm, R. Johnson, and J. Vlissides. 1995. *Design Patterns: Elements of Reusable Object-Oriented Software*. Boston, Massachusetts, USA. Addison-Wesley Longman Publishing Co., Inc.

[13] Palsberg, J. and C. B. Jay. 1998. The Essence of the Visitor Pattern. *The 22nd International Conference on Computer Software and Applications*. Vienna, Austria. 19-21 August 1998. 9-15.

[14] Wohlin, C., P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén. 2000. *Experimentation in Software Engineering: An Introduction*. MA, USA. Kluwer Academic Publishers.

[15] Elish, M. and M. Mohammed. 2015. Quantitative Analysis of Fault Density in Design Patterns: An Empirical Study. *Information and Software Technology*. 66(2015): 58-72.

[16] Sfetsos, P., A. Ampatzoglou, A. Chatzigeorgiou, I. Deligiannis, and I. Stamelos. 2014. A Comparative Study on the Effectiveness of Patterns in Software Libraries and Standalone Applications. *The 9th International Conference on Quality of Information and Communications Technology*. Guimaraes, Portugal. 23-26 September 2014. 145-150.

[17] Jeanmart, S., Y. G. Gueheneuc, H. Sahraoui, and N. Habra. 2009. Impact of the Visitor Pattern on Program Comprehension and Maintenance. *The 3rd International Symposium on Empirical Software Engineering and Measurement*. Florida, USA. 15-16 October 2009. 69-78.

[18] Khomh, F. and Y. G. Gueheneuce. 2008. Do Design Patterns Impact Software Quality Positively? *The 12th European Conference on Software Maintenance and Reengineering*. Athens, Greece. 1-4 April 2008. 274-278.

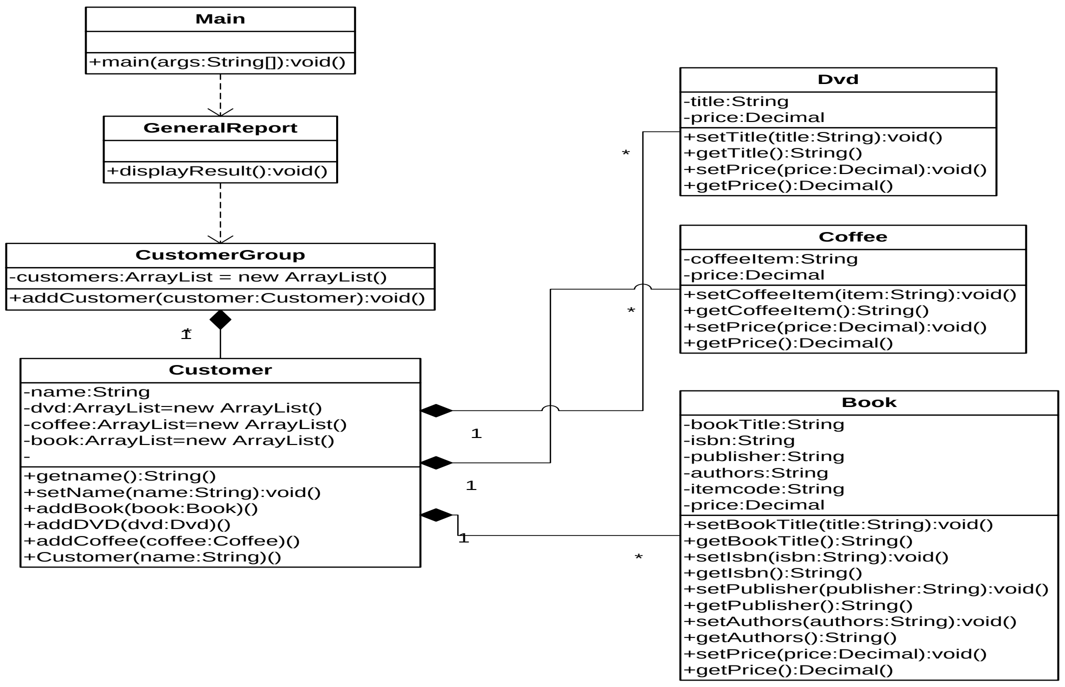
[19] Razali, R., C. F. Snook, and M. R. Poppleton. 2007. Comprehensibility of UML-Based Formal Model: A Series of Controlled Experiments. *The 1st ACM International Workshop on Empirical Assessment of Software Engineering Languages and Technologies*. Atlanta, USA. 5-9 November 2007. 25-30.

[20] Dunlop, W. P., J. M. Cortina, J. B. Vaslow, M. J. Burke. 1996. Meta-Analysis of Experiments with Matched Groups or Meta-analysis. *Psychological Method*. 1(2): 170-177.

Appendix A - Diagrams

Diagram 2 - We have created these class diagrams from a reporting software module used in a store selling DVDs, VCDs and coffee to customers. The reporting module provides statistics about the customers who are members of the store. Note: Diagram 1 is shown in Figure 1.

Diagrams 3 and 4 - We have created these diagrams from a software module used in a hospital. This module provides information for modeling outpatients' behaviors. In this system, we have classified the patients into three groups based on age range: 0 – 15 years, 16 – 45 years and over 45 years.



Main
+main(args:String[]):void()
GeneralReport
+displayResult():void()
CustomerGroup
-customers:ArrayList = new ArrayList()
+addCustomer(customer:Customer):void()
1
Customer
-name:String
-dvd:ArrayList=new ArrayList()
-coffee:ArrayList=new ArrayList()
-book:ArrayList=new ArrayList()
-
+getname():String()
+setName(name:String):void()
+addBook(book:Book)()
+addDVD(dvd:Dvd)()
+addCoffee(coffee:Coffee)()
+Customer(name:String)()
Dvd
-title:String
-price:Decimal
+setTitle(title:String):void()
+getTitle():String()
+setPrice(price:Decimal):void()
+getPrice():Decimal()
Coffee
-coffeeItem:String
-price:Decimal
+setCoffeeItem(item:String):void()
+getCoffeeItem():String()
+setPrice(price:Decimal):void()
+getPrice():Decimal()
Book
-bookTitle:String
-isbn:String
-publisher:String
-authors:String
-itemcode:String
-price:Decimal
+setBookTitle(title:String):void()
+getBookTitle():String()
+setIsbn(isbn:String):void()
+getIsbn():String()
+setPublisher(publisher:String):void()
+getPublisher():String()
+setAuthors(authors:String):void()
+getAuthors():String()
+setPrice(price:Decimal):void()
+getPrice():Decimal()
*
*
*
1
1
1

Figure 9 Diagram No.2

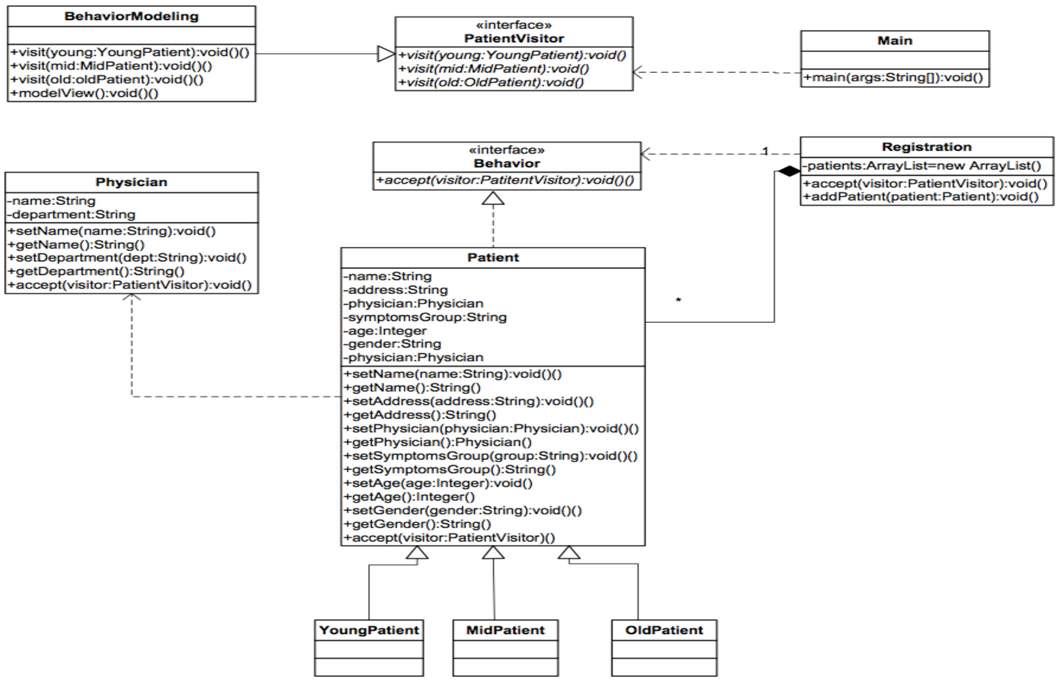

BehaviorModeling
+visit(young:YoungPatient):void()()
+visit(mid:MidPatient):void()()
+visit(old:oldPatient):void()()
+modelView():void()()
«interface» PatientVisitor
+visit(young:YoungPatient):void()
+visit(mid:MidPatient):void()
+visit(old:OldPatient):void()
Main
+main(args:String[]):void()
«interface» Behavior
+accept(visitor:PatitentVisitor):void()()
Registration
-patients:ArrayList=new ArrayList()
+accept(visitor:PatientVisitor):void()
+addPatient(patient:Patient):void()
1
*
Physician
-name:String
-department:String
+setName(name:String):void()
+getName():String()
+setDepartment(dept:String):void()
+getDepartment():String()
+accept(visitor:PatientVisitor):void()
Patient
-name:String
-address:String
-physician:Physician
-symptomsGroup:String
-age:Integer
-gender:String
-physician:Physician
+setName(name:String):void()()
+getName():String()
+setAddress(address:String):void()()
+getAddress():String()
+setPhysician(physician:Physician):void()()
+getPhysician():Physician()
+setSymptomsGroup(group:String):void()()
+getSymptomsGroup():String()
+setAge(age:Integer):void()
+getAge():Integer()
+setGender(gender:String):void()()
+getGender():String()
+accept(visitor:PatientVisitor)()
YoungPatient
MidPatient
OldPatient

Figure 10 Diagram No.3

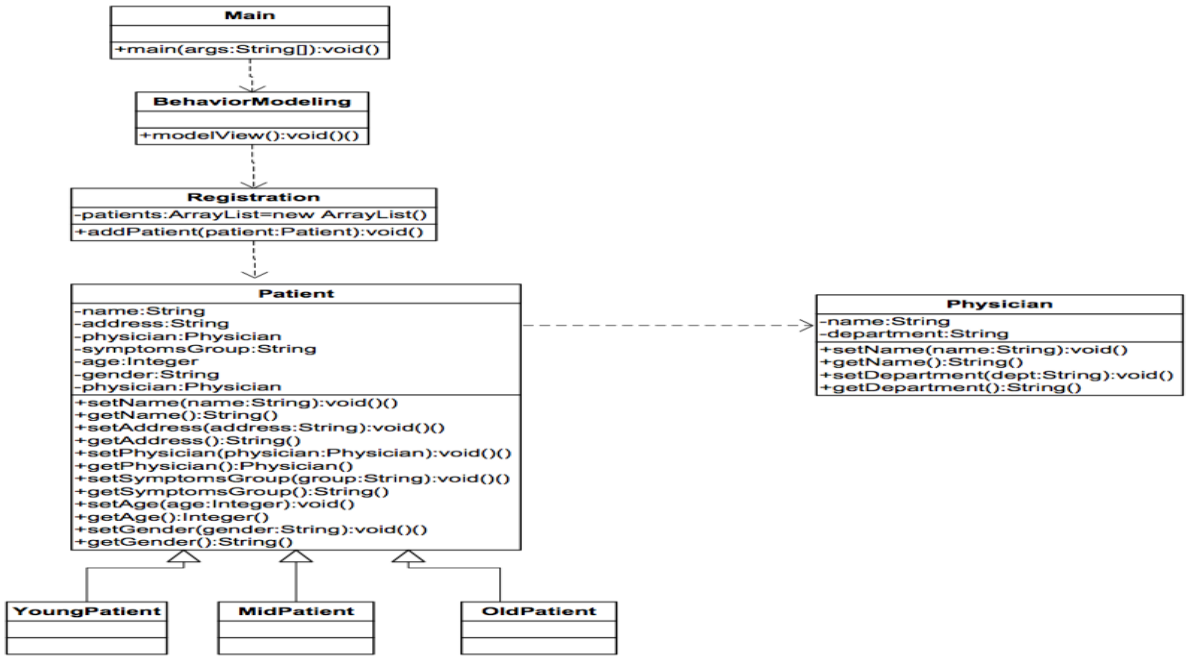

Main
+main(args:String[]):void()
BehaviorModeling
+modelView():void()()
Registration
-patients:ArrayList=new ArrayList()
+addPatient(patient:Patient):void()
Patient
-name:String
-address:String
-physician:Physician
-symptomsGroup:String
-age:Integer
-gender:String
-physician:Physician
+setName(name:String):void()()
+getName():String()
+setAddress(address:String):void()()
+getAddress():String()
+setPhysician(physician:Physician):void()()
+getPhysician():Physician()
+setSymptomsGroup(group:String):void()()
+getSymptomsGroup():String()
+setAge(age:Integer):void()
+getAge():Integer()
+setGender(gender:String):void()()
+getGender():String()
Physician
-name:String
-department:String
+setName(name:String):void()
+getName():String()
+setDepartment(dept:String):void()
+getDepartment():String()
YoungPatient
MidPatient
OldPatient

Figure 11 Diagram No.4

Appendix B – Questions

Questionnaire (A)
Name: ____________________
Diagram No: ( ) 1 ( ) 2
Write down the starting time (HH:MM:SS):____________________

1. How does the GeneralReport show the detail of the Customer(s)?
2. How does the GeneralReport show the detail of the Coffee(s) for each Customer?
3. Can the GeneralReport show the total price of product(s)? Please give the justification.
4. Suppose the store need to remove the Book from the existing system? Do we need to change the GeneralReport? Please give the justification.
5. Suppose the store need to add more information in Customer from the existing system? Do we need to change the GeneralReport? Please give the justification.
6. Can the Coffee know the Customer who is belong to? Please give justification.

Write down the ending time (HH:MM:SS):____________________

Questionnaire (B)
Name:____________________
Diagram No: ( ) 3 ( ) 4
Write down the starting time (HH:MM:SS): ____________________

1. How does the BehaviorModeling show the behavior of patients?
2. How does the BehaviorModeling show the detail of the MidPatient?
3. Can the BehaviorModeling show the information of Physician for each patient?
4. Suppose the hospital need to remove the OldPatient group from the existing system. Do we need to change the BehaviorModeling? Please give the justification.
5. Suppose the hospital need to add more information in Patient from the existing system. Do we need to change the BehaviorModeling? Please give the justification.
6. Can the Physician know the Patient who has been treated? Please give justification.

Write down the ending time (HH:MM:SS): ____________________

Appendix C – Survey Questions (Post-Study)

1. Have you ever used the UML class diagram?
   ( ) Yes ( ) No
2. Please rate the easiness of reading the diagram no. 1 (or 2)
   ( ) Very Difficult ( ) Difficult ( ) Neutral ( ) Easy ( ) Very Easy
3. Please rate the easiness of reading the diagram no. 3 (or 4)
   ( ) Very Difficult ( ) Difficult ( ) Neutral ( ) Easy ( ) Very Easy
4. For the Questionnaire-A, how confident that you have correctly answered the questions?
   ( ) Very Unsure ( ) Unsure ( ) Neutral ( ) Sure ( ) Very Sure
5. For the Questionnaire-B, how confident that you have correctly answered the questions?
   ( ) Very Unsure ( ) Unsure ( ) Neutral ( ) Sure ( ) Very Sure
6. Please rate the Visitor pattern experience before performing the given tasks

( ) No experience	( ) Learned in a class or from books
( ) Used on a class project	( ) Used on several class project
( ) Used on a real industry project	