Detecting the correct design pattern for enhanced maintainability - MPT

Ashutosh kumar Srivastava
MCA, University of Allahabad

Abstract— The effect of design patterns on the software maintainability is governed by different factors such as pattern size, prior expertise of the developer with pattern and the most important quality attributes that must achieved by pattern , and before all of these is fitting the pattern to a certain design problem . In the authors have created a decision support tool that helps the developer to choose between three of GoF design patterns and equivalent alternative design solutions, it calculates metrics scores of each solution based on the system size, then it presents where a design solution is getting better than another with respect to several quality attributes.

This research study solely emhasizes on which design pattern is useful to improve the maintainability of the software on the basis of the Metrics as measurement of maintainability and we will be carrying out our research on this ground.

Index Terms— Design patterns, Maintainbility, Gang of Four, Maintainbility Predictors, Architectural Patterns, Idioms Pattern, Software Metrices.

1 INTRODUCTION

THE design pattern is a general reusable solution to a commonly occurring problem in software design. It can be defined as a description or template for how to solve a problem that can be used in many different situations .

There are three main types of design patterns that are architectural patterns, Gang of Four (GoF) design patterns and idiom patterns.The archirectural pattern constitutes of the whole system where the modules are taken as a single unit and the entities interact through the object oriented paradigm.the GoF pattern published a book name **Design Patterns - Elements of Reusable Object-Oriented Software** according to which the patterns are divided as following way- Creational pattern which states for the way of creation of object while hiding the creation logic from the client. Another one is the Structural pattern in which the concern is with the object and class creation via inheritance and interfaces, and the last one is the Behavioural pattern which deals with the communication between the objects, another ones is the **J2EE** patterns which are exclusively concerned with the presentation tier. The last is the idiom patterns which is solely dedicated to the coding phase. Each architectural pattern can be cited as a design pattern but not every design pattern can be termed as a architectutal pattern.

These design patterns tends to ease the developers the ways to handle each and every module in a similar manner so no need to think of a newer ways to handle any module.

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

2 RELATED WORK

The study of design pattern has been subjected to empirical evaluation. Many research has been carried out to predict the ways which can detect the correct design pattern which can actually enhance the maintainability of codes but no reaseach has been judged as that much successful in order to be called substantial work for the same. Surveys has been carried out, naïve programmers were hired to detect the performance of these design patterns on certain piece of codes. In some case, some of the patterns were proving fruitful to the case whereas in some case, the another one gave better result and it's an undeniable truth about the design patterns that no design pattern can actually be called out as a supreme pattern to be employed in each and every case.

Its quite impossible also to find out a tool which can actually check the performance of the architectutral patterns on codes as different scenarios are there in the wordly problems and each one of them needs to be handled differently, so most of the work was restricted to only Gang of Four pattern and that's too only Behavioural pattern. The one considerable research has ben carried out by the Prechelt et al.

They conducted an experiment called PatMain by comparing the maintainability of two implementations of an application, one using a design pattern and the other using a simpler alternative. They used four different subject systems in same programming language. They addressed five patterns: Decorator, Composite, Abstract Factory, Observer and Visitor. The researchers measured the time and correctness of the given maintenance tasks for

professional participants. They found that it was useful to use a design pattern but in case where simple solution is preferred, it is good to follow the software engineer common sense about whether to use a pattern or not, and in case of uncertainty, it is better to use a pattern as a default approach. A thorough understanding of specific design patterns is often helpful for program maintenance.

Juristo and vegas conducted yet another experiment in replication to the Patmain experiment in which they considered the decorator patterns, the abstract factory pattern and composite pattern and they took 8 programming students and made them to apply them on certain piece of codes and the time was only the variables on which the study was based and the result came out was absolutely inconsistent with respect to the original study and experiment. They found that system design pattern was less maintainable.

Naanthaamorn phong and caver also replicated the Patmain experiement and they included the visitor pattern, decorator pattern, observer pattern and composite pattern to carry out the same research and they took 10 engineering students to apply them on certain piece of codes. The result of the study was different from those of the original study. They found the design patterns were not able to enhance the maintainability and the understandability of the codes.

Krein et al performed the replication of the Patmain experiment and they included two systems with different languages and they applied composite pattern, visitor patern and the abstract factory pattern on certain piece of codes, they found that on modifying certain things on a versions, design pattern based codes were found at errors as compared to the system with non design patterns based codes.

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

Zhang and budgen conducted a review of the literatures that were given by many eminent persons on the empirical study of the knowledge of the Gang Of Four patterns and they went through it thouroughly. They facilitated their analysis on the basis of some experience reports that stated about the design patterns and their applications using some less rigorous observational forms. They found the grounds on which they can lay this statement that design patterns have some effect on the maintainability of codes but they cant found any solid base of their selection as for which design pattern is able to increase the maintainability of the codes more than other patterns.

Ali and Elish performed the literature survey and they included in their study the impact of the Gang of Four design patterns on four different attributes that were maintainablility, evaluation, performance and faultproneness. The results show that in general, the impact of design patterns on maintainability, evolution and change proneness is negative. For performance, the number of studies that addressed performance and the number of covered patterns make it difficult to draw a conclusion. Finally for faultproneness, the results are different from one study to the other, thus it is difficult to make a decision regarding their impact.

Hsueh carried out an analytical assessment to help the programmers to study and inspect the correctness of these design patterns and their efficiency. They also proposed two different measures: Occasion and effectiveness analysis to study some well known open source systems. They defined their context and their anticipated changes and then checked whether they held up to the expectations. Their conclusion was that although design patterns can be misused their effectiveness is maintained and they prove to be useful at early or later stages of maintenance.

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

Nadia et al conducted one study that created a tool which provide some guidelines and some recommendation rules that can help programmers decide whether the intention imposed for the design patern is acheived or not. The tool allows the designer to draw a design fragment, present the problem then re-phrases the problem in order to obtain the intention of a certain pattern. Then, it explores the candidate solutions by filtering patterns that meet the intentions through the use of recommendation rules.

3 PROBLEM STATEMENT

Which design pattern is more maintainable with respect to another in a given problem statement and under what conditions?

4 PROPOSED SOLUTION

Until now there is no study or the research has been done in order to carry out the check on the maintainability prediction of the design patterns as which of them has stronger effects on it and which of them weakens the maintainability. It is firmly said above that no tool can actually classify whole of the patterns and detect the efficiency of these on a given set of problems, codes and statements but from the GoF patterns one can predict the working of some of the design patterns on a given set of codes and scenarios. The effect of these patterns can be judged on many of the factors such as system size, the prior information of the ines of codes and the imposed pattern but there are certain set of maintainability predictors on the basis of which the prediction becomes quite easy with respect to many different other factors that are there.

Ampatzoglou[4] have created a decision support tool that helps the developer to choose between three of *GoF* design patterns and equivalent alternative design solutions, it calculates metrics scores of each solution based on the system size, then it presents where a design solution is getting better than another with respect to several quality attributes. This paper have proposed a new version of this tool that aims to compare the maintainability of *GoF* design patterns with each other based on the maintainability predictors.

The design patterns that are considered for this maintainability predictor tools are Gang of four patterns. The inputs of the tool will be the particiapating classes, and on modifying the interfaces participating or introducing the new or naïve client, the no. of participating classes is the most common parameters taken to study the maintainablilty of the codes. The major axis on which the tool will be studied is the no.of refined abstract classes, no.of concrete implementer classes, no.of new clients and the new methods and attributes that were introduces for the chosen pattern. In earlier studies mainly two of the given axis were chosen as no.of concrete implementer classes and no.of refined abstract classes and the tool was given to study the maintainability on the basis of some maintainability predictor elements also known as software metrices. Taking these two axis as parameters they chose no.of metrices that they were interested in and they chose the pattern and calculated he average of the metrices scores on the design patterns. They summed up the grand total of the metric scores and pattern with the higher no.of least metric value or he metric score with the least sum of the metrics is chosen as the best design pattern.

But no tabular explanation of the tool was given in the earlier study so I have explained the maintainability predictor tool or MPT with a rough explanation of carrying outthe calculation of the metric scores and thsts too on a most easier and abstract level. I have given the explanation which can be brought into consideration for engineering students who are just mere beginners.

4.1 LIST OF THE MAINTAINABILITY PREDICTORS

Metrics descriptions

- DIT - Depth of the inheritance tree (=inheritance level number of the class, 0 for the root class).
- MPC - Message-passing couple (=number of send statements defined in the class).
- NOC -Number of children (=number of direct subclasses that the class has).
- RFC -Response for a class (=total number of local methods and the number of methods called by local methods in the class).
- DAC - Data abstraction coupling (=number of abstract data types defined in the class).
- WMPC - Weighted method per class (=sum of McCabe‟s cyclomatic complexity of all local methods in the class).
- NOM -Number of methods (=number of local methods in the class).
- SIZE1 -Lines of code (=number of semicolons in the class).
- SIZE2 -Number of properties (=total number of attributes and the number of local methods in the class).

4.2 PROPOSED TOOL

- The decision support tool has been designed in to choose any three design patterns from creational pattern of (gof), it calculates metrics scores of each solution based on the system size.
- then it presents where a design solution is getting better than another with respect to several quality attributes from list of maintainability predictors or metrics.
- The software design with maximum number of minimum metric scores is judged as a software design which increases the maintainability of codes.
- The proposed tool aims to help the designer/developer to choose the appropriate design pattern that produces more maintainable system. The input of the tool is the pattern under investigation and the estimated pattern size which is number of refined abstractions classes (n) and number of concrete implementers classes (m). The functional architecture of proposed tool is shown in figure 1, the user selects the pattern he wants to examine then selects the metrics he is interested in and finally defines the (n) and (m) for the pattern.

The tool retrieves all patterns that describe equivalent functionality from a repository of patterns, and then calculates the mathematic equations of selected metrics for each equivalent pattern. The tool displays the results in two phases: first phase indicates the average metric scores for each pattern in the given range of (n) and (m), and the second phase determines which pattern produces „best‟ re-

sults i.e. has the higher count of lower metric values then consider as more maintainable.

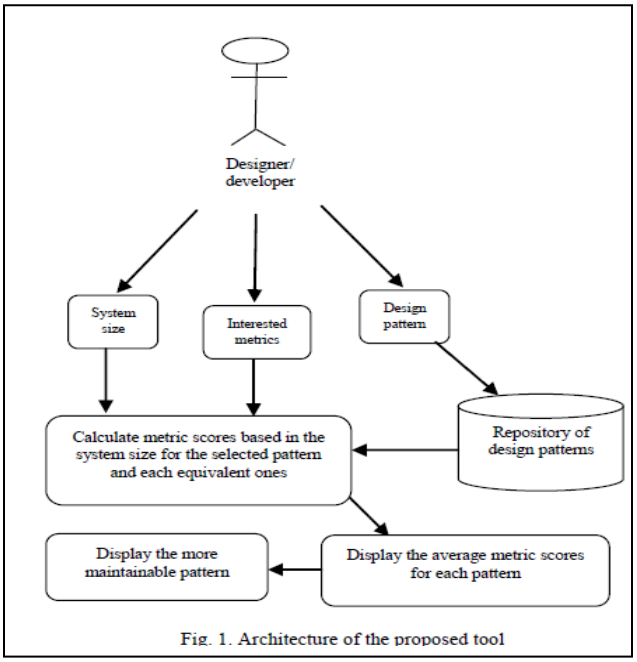

Fig. 1. Architecture of the proposed tool

.

4.3 EXPLANATION

Suppose we have taken three design patterns who have the value of the concrete implementer classes and no.of refined abstract classes as pattern1(1,5), pattern2(0,6) and pattern3(0,4) and the metrics be chosen as metric 1, metric 2, and metric 3 so the table that would be drawn for the same will be like this for a given module-

TABLE 1
EXPLANATION OF THE TOOL IN TABULAR FORM

Patterns(n,m)	Pattern 1(1,5)	Pattern 2(0,6)	Pattern 3(0,4)
Metric 1	1	1	2
Metric 2	1	0	1
Metric 3	1	0	0

On adding the total of these metrics value, the total of pattern 1 comes out to be 3, the total of pattern 2 comes out to be 2 and the grand total of pattern 3 comes out to be 3. So the least sum value of the three patterns included is of pattern 2 so pattern 2 is judged as the most maintainable pattern.

5 VALIDATION OF THE TOOL

The survey was conducted online for the engineering students who were asked to take the simpler design patterns that are given above and apply the MPT on the metrics according to their choice. The chosen design patterns were abstract factory patern and the factory pattern.The result that came out was that factory pattern came out to be more maintainable.

TABLE 2
DESIGN PATTERN UNDER CONSIDERATION

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

The total students involoved in he survey were 50 and out of those students 20 said that the tool is useful 15 students said that tool the working is feasible but specifications and modifications can be done over the tool, 10 students said the tool lack the logic and 5 said they couldn't understand the working of the tool.

6 CONCLUSION

The authors proposed a solution to evaluate the effect of design patterns on software maintainability. This solution is simulated by a tool that measures the maintainability of each pattern by some relevant metrics with regard the system size.

The future work on the maintainability is required as for the design patterns on applying the system lines of codes as metrics creates ambiguity and the tool should accept the system size automatically and on the basis of rest of the metrics the tool can carry out to take the same calculations as it is supposed to carry out. This will enhance the specifications of the design patterns and the accuracy of the tool will be enhanced automatically.

7 REFERENCES

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

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

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

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

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

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

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

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

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

Ashutosh kumar Srivastava is currently pursuing Master degree course in Computer Applications from University of Allahabad, India.
Mobile No- 08382069571
Email id-developerashutosh03@gmail.com