On Use of Design Patterns in Empirical Assessment of Software Design Quality

Md. Abul Khaer, M.M.A. Hashem, Md. Raihan Masud
Department of Computer Science and Engineering
Khulna University of Engineering and Technology (KUET)
Khulna 9203, Bangladesh
khaer_mohammad@yahoo.com, mma_hashem@hotmail.com, raihan_masud@ieee.org

Abstract

This paper describes the results of a statistical analysis where the impact of design patterns on object oriented software system is experimentally evaluated. We used GoF (Gang of Four) design patterns as our assessment criteria. For empirical analysis data is collected from nine different size open source java projects. The results obtained in this experiment show how design quality of software systems can be affected by proper usage of design patterns. In this paper, we also propose an approach for measuring design quality of software using design patterns. We validate our proposed approach against quality metrics based approach.

I. Introduction

Software system builders increasingly recognize the importance of exploiting design knowledge in the engineering of new systems. It is widely argued that low design quality of software causes systems to exhibit low maintainability, low reuse, high complexity and faulty behavior. Object oriented (OO) systems frequently need modification, extension, detection and correction. So, for OO systems poor design quality can make this entire tasks complex. The pattern community promises that using design patterns enhances design quality.

Several models have been proposed to estimate quality of software systems. They are based on the prediction of fault proneness of software module [1,2,3], on detection of anti pattern [4] which is known to be bad coding practices, based on OO metrics [5,6] and visualization technique [7]. We studied [8][9] for impact of patterns on OO systems. We found that no empirical analysis has been performed to measure quality of OO systems using patterns. This issue remains in the backbench. We tried to focusing on that area. In practice, measurement of software quality is not easy. So, we did an empirical study. We validate our proposed formula against OO quality metrics.

The rest of the paper is organized is as follows: Section II describes experimental investigation. Section III depicts experimental results and comparison with other methods. Finally section IV concludes the paper.

II. Description Of The Experiment

A. Procedure

We split our working procedure on several steps. Firstly, we select some open source projects and we select PINOT [10] after scrutinizing a lot of tools for mining design patterns from projects. Secondly, we select two CK metrics and one traditional metric to validate our experiment. Finally, we propose an approach for measuring Design Quality Level (DQL) to perform our experiment and validate it against current metric based approach. The whole workflow is shown sequentially in Figure 1.

B. Experimental Materials

*1) Open source projects: W*e select some open source projects to show that use of design patterns in software design enhances software quality. They are Java swing, JHot Draw, Java util, Fitnesse, JDOM, Deputy, Card Game, Network Protocol, Mobile Game. Table 2 lists the open source projects' description with design quality level assumption.

2) Design Patterns: The design patterns are language-independent strategies for solving common object-oriented design problems. Quality of software depends on various facts such as reusability, modularity,

978-1-4244-1692-9/08/$25.00 ©2008 IEEE

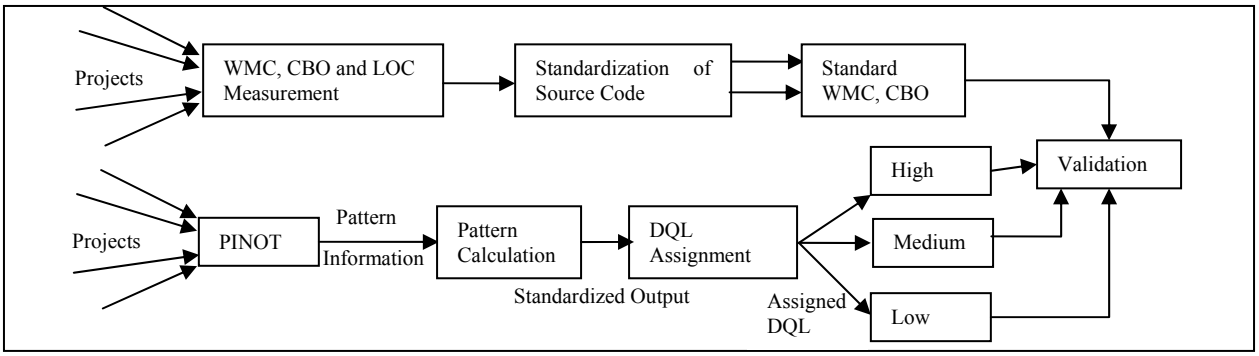

Figure 1: Workflow of our proposed approach

TABLE 1: OPEN SOURCE PROJECTS WITH DIFFERENT DESIGN QUALITY LEVEL

Projects	Description
Java Swing	Java class library, surely designed with high reusability
JHot Draw	Designed by Erich Gamma, Design Pattern Guru
Java Util	Java class library, surely designed with high reusability
Fitness	Acceptance Testing Framework by Robert C. Martin CEO, Object Mentor Inc, writer arena of OO Design
JDOM	XML, Jason Hunter, author , "Java Servlet Programming", JDOM received JSR-102 by Java Community Process
Deputy	A plug in for Maven Project, Matthious Barbach, expert in Web Programming
Network Protocol	Layered Implementation of Computer Networks, lab work by senior undergrad students
Mobile Game	Mobile Device game in J2ME, junior undergrad project, won a National Software Fair Award
Card Game	Desktop Game in Java, junior undergrad project

abstractness, understandability, maintainability etc. Design pattern guarantees reusable design by avoiding creating objects directly, avoiding dependencies on specific operations, avoiding algorithmic dependencies and avoiding tight coupling. It makes software flexible, modular, and understandable and makes the maintenance task uncomplicated. They are pre-designed and tested solutions of fundamental design problem leading advantage for developers. Communities' already using design patterns as it offers extensive reusability, easy maintainability, more modularity etc. We count total pattern in java swing 1.4.3 and java swing 1.5.6 using a tool PINOT [10]. Results shown in Table 2 indicates usage increase of design patterns. This scenario is not only for java swing but also for all standard object oriented systems. So, counting design patterns in OO software systems can be used to explore their design quality level.

TABLE 2: TOTAL PATTERNS IN TWO SWING VERSION

J a v a s w i n g	Java swing 1.5.3
1258	1825

3) Design Pattern Mining Tool: There ware several tools available for mining design pattern from java source code e.g FUJABA, PTIDEJ, PINOT. We used the tool PINOT [10] which is faster, more accurate, and targets more patterns than other existing pattern detection tools. PINOT is a modification of Jikes which is an open source C++ Java compiler. Other tools fail to identify some patterns though they provide recognition for them. But PINOT can identify all the patterns successfully for which it has definition.

4) OO Metrics: To evaluate accuracy of our proposed approach we select two OO metrics Coupling Between Objects (CBO) and Weighted Method Per Class (WMC) and a traditional metric LOC which measures lines of codes. The WMC is a count of the methods implemented within a class or the sum of the complexities of the methods (method complexity is measured by cyclomatic complexity). Cyclomatic complexity is software metric that provides a quantitative measure of the logical complexity of a program. The number of methods and the complexity of the methods involved is a predictor of how much time and effort is required to develop and maintain the class. This metric measures usability and reusability. CBO is a count of the number of other classes to which a class is coupled. Excessive coupling is detrimental to modular design and prevents reuse. The more independent a class is, the easier it is reused in another application. The larger the number of couples, the

higher the sensitivity to changes in other parts of the design and therefore its maintenance is more difficult.

III. Experimental Results

A. Patterns per 10 Classes (P10C) Approach

We mined design patterns by PINOT [10] which runs under Linux environment by the IBM compiler Jikes. It outputs total individual patterns in those projects. Still there is no tool which can find all the patterns. PINOT detects all the GoF patterns that have concrete definitions driven by code structure or system behavior. We apply pattern mining operation on nine selected projects. Figure 2 shows total patterns after standardization of source code.

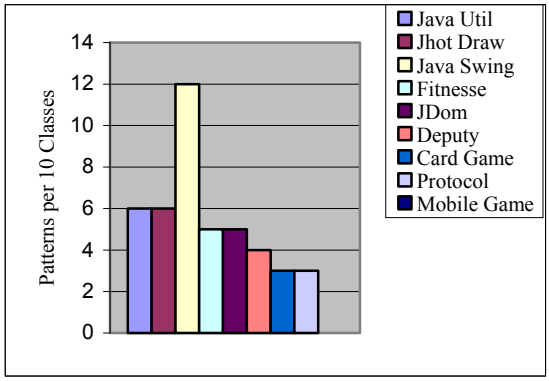

Figure 2: Total patterns after standardization of source code

After configuring the environment we provide open source projects as input to the PINOT toolkit. It outputs existence of patterns in the projects. It is typical that a project comprising 100 classes (large project) has more patterns than a project having 50 classes (small project). So, we have to standardize it to a common threshold value otherwise it will show erroneous results. For ensuring quality level we select patterns per 10 classes ($P10C$) as our threshold value. The Table 3 uses the terms as following meaning:

NoC = No of Processed Class
TP = Total Pattern Found
$P10C$ = Pattern per 10 Classes

The formula for calculating $P10C$ is

$$P10C = \left\lceil \left(\frac{\text{Total Pattern in Project}}{\text{Number of Class Processed}} \right) * 10 \right\rceil = \lceil (TP/NoC) * 10 \rceil \quad (1)$$

A simple example of converting to $P10C$ is shown here. For java util, TP =59, NoC =101

$$P10C = \left\lceil \frac{59}{101} * 10 \right\rceil = 6$$

However, we defined a standard of DQL (Design Quality Level) BASED ON THIS $P10C$. WE ALSO PROPOSED three categories which are C_H, C_M, C_L which represents DQL high, medium and low. So for a system, its categorization should be as follows:

if $P10C \geq 6$ then C_H
if $P10C \geq 4$ and ≤ 5 then C_M
if $P10C \geq 0$ and ≤ 3 then C_L

Table 3 shows the measured design quality and categories according to above classification where term C stands for a particular category.

TABLE 3: DQL OF PROJECTS BASED ON $P10C$

Projects	*NoC*	*TP*	*P10C*	***DQL***	***C.***
Java Util	101	59	6	High	C_H
JHot Draw	123	74	6	High	C_H
Javax Swing	1485	1825	12	High	C_H
Fitnesse	255	134	5	Medium	C_M
JDOM	40	20	5	Medium	C_M
Deputy	120	47	4	Medium	C_M
Card Game	16	5	3	Low	C_L
Protocol	7	2	3	Low	C_L
Mobile Game	0	0	0	Low	C_L

B. Mathematical Model

For an object oriented system, the mathematical model of our approach is described here. A new term p is added which indicates individual GoF patterns. Meaning of other terms is remained as before. This model summarizes calculation of patterns per 10 classes ($P10C$) and assignment of design quality level (DQL) in a short.

Mathematically measurement of $P10C$ is summarized as follows-

P_i = GoF Design Patterns where i= 1 to 23

$$TP = P_1 + P_2 + P_3 + \ldots\ldots\ldots\ldots\ldots + P_{23} = \sum_{i=1}^{23} P_i \quad (2)$$

$$P10C = \frac{P_1 + P_2 + P_3 + \ldots\ldots\ldots\ldots\ldots + P_{23}}{NoC} * 10 = \frac{\sum_{i=1}^{23} P_i}{NoC} * 10 \quad (3)$$

DQL Assignment

DQL = High if $P10C$ ranges from 6 to above

DQL = Medium if $P10C$ ranges from 4 to 5

DQL = Low if $P10C$ ranges from 0 to 3.

C. Programmatic Model

A programmatic model of our approach is described here. This model summarizes calculation of patterns per 10 classes ($P10C$) and assignment of design quality level (DQL) in a short-

P_i = *GoF Design Patterns where i= 1 to 23*

S_j = *OO systems where j= 1 to m*

foreach system S_j *in the experiment*

TP = P10C = NoC = 0

foreach GoF pattern P_i

TP= getTotalPattern(S_j*)*

NoC= getTotalProcessedClass(S_j*)*

*P10C = ceil(TP/NoC)*10*

DQL= getDQL (P10C, S_j*)*

getTotalPattern(S_j*)*

return total pattern found in S_j by PINOT

getTotalProcessedClass(S_j*)*

return total processed class for system S_j

getDQL (P10C, S_j*)*

if $P10C \geq 6$ then *DQL= High*

if $P10C \geq 4$ and ≤ 5 then *DQL = Medium*

if $P10C \geq 0$ and ≤ 3 then *DQL = Low*

return DQL

D. Metric Value Calculation

We measure WMC and CBO from the open source projects by a metric tool *metrics 1.3.6*[11]. This tool is an open source tool and a plug in of Eclipse. OO metrics value swings against projects size. If we pay no attention to it and just consider metrics value disregarding size it will cause an erroneous calculation. So for different size projects we have to consider a standard volume. We propose here a standard 10K (ten kilo lines) code. Measured value is shown in Table 4.

An example of value against 10K is shown below-

For java util, $LOC = 20104$

$$CBO = \frac{25}{20104} * 10K = 12.44;$$

$$WMC = \frac{20}{20104} * 10K = 9.95$$

TABLE 4: CK METRIC VALUE IN PROJECTS.

Projects	LOC	Normal		10K Code	
		CBO	WMC	CBO	WMC
Java Util	20104	25.00	20	12.44	9.95
JHot Draw	21926	15.17	15	6.91	6.84
Java Swing	134168	32.25	26	2.30	1.93
Fitnesse	7892	14.34	8	19.67	10.14
JDom	14746	21.14	38	14.33	25.77
Deputy	2786	23.50	17	84.35	61.02
Card Game	699	29.00	50	414.00	715.31
Network Protocol	1189	25.00	10	210.26	84.10
Mobile Game	466	6.00		128.76	600.85

High value if CBO means poor design. On the other hand WMC ensure code complexity, design usability and reusability. We have to keep it low. Otherwise abstractness of design will go down.

TABLE 5: COMPARISON OF DQL WITH CK METRICS APPROACH AND DESIGN PATTERN BASED APPROACH

SL. No.	Projects	*DQL* by pattern	*WMC*	*CBO*
1	Java Util	High	9.95	12.44
2	JHot Draw	High	6.84	6.91
3	Javax Swing	High	1.93	2.39
4	Fitness	Medium	10.14	19.67
5	JDOM	Medium	25.77	14.33
6	Deputy	Medium	61.02	84.35
7	Card Game	Low	715.31	414
8	Net. Protocol	Low	84.10	210.26
9	Mobile Game	Low	600.85	128.76

E. Result Analysis & Comparison

We compare quality level by design pattern based approach and OO metric based approach. In both cases we did standardization by class and lines of code to remove dissimilarity between projects. This standardization makes our approach more precise, reliable and clearer. For this standardization we can measure quality of small to large projects or module by module. Table 5 shows a comparison of design quality level between our pattern based approach and metrics based quality level. The graphical representation of

Table 5 is shown in Figure 3. From Figure 3, it is observed that projects having high DQL have lower WMC and CBO value than others, projects having medium DQL have higher WMC, CBO value than DQL high projects but lower than others and projects having low DQL have worst WMC, CBO value. Our *P10C* approach found DQL of java util, swing and jhotdraw as *high*.

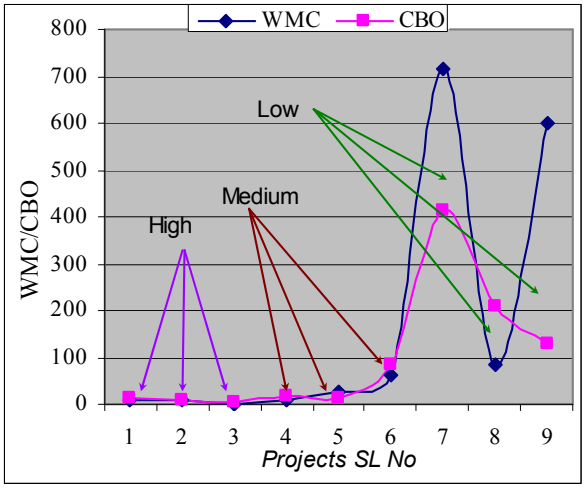

Figure 3: Pattern vs Metric Graph

On the other hand metrics based approach shows their *WMC* and *CBO* value better than all others. For Fitness, JDOM, Deputy *P10C* approach finalizes their DQL as *medium*. Metrics based approach calculate their *WMC* and *CBO* value better than students projects but worse than util, swing and jhotdraw. For students projects pattern based approach encompasses their DQL as *low*, *medium, low* and metric based approach found their *WMC* and *CBO* value worst than all others. Our approach based on design pattern which fulfill all the necessity of good OO design. So, software professional can use our procedure for measuring design quality of systems. They don't need to measure OO metric because OO metric cannot describe OO systems properly. This experimental result supports our approach strongly and indicates that this can be a new way of software design quality assurance.

IV. Concluding Remarks

In this paper we have presented an empirical study for OO systems quality analysis and proposed an approach based on design pattern. We carry out a comparison with OO metric approach which ensures validity of our proposal. To the way of this endeavor we make standardization of source code to remove dissimilarity between small and large projects. Our semi-automatic approach is a good compromise between design patterns and software quality which can be useful and efficient. It can estimate design quality of small to large systems. OO quality metrics can't describe OO systems fully; they can reflect some aspects of the design quality such as: complex methods/classes, package structure and the abstraction in a system. But object-oriented design is much more than that which is fulfilled by design patterns. Our Future work includes developing strong algorithms so that all the GoF patterns can be identified. We plan to develop a toolkit. Then our experimental result will be more precise. Finally we conclude that the experiment we conducted can be helpful for communities, quality assurance managers, programmers and to all practitioners.

References

[1]. L. C. Briand, W. L. Melo, and J. Wust, "Assessing the applicability of fault-proneness models across object-oriented software projects", *IEEE Transactions on Software Engineering*, vol. 28, no. 7, pp. 706-720, July 2002.

[2]. Shi Zhong, M. Khoshgoftaar, and Naeem Seliya, "Expert-Based Software Measurement Data Analysis with Clustering Techniques", Accepted to IEEE Intelligent Systems, Special Issue on Data & Information Cleaning & Preprocessing, 2004.

[3]. Yuming Zhou and Hareton Leung, "Empirical Analysis of Object-Oriented Design Metrics for Predicting High and Low Severity Faults," *IEEE Trans. Software Eng.*, vol. 32, no. 10, pp. 771-789, Oct. 2006.

[4]. W.J. Brown, R.C. Malveau, H.W. McCormick, III, and T.J. Mowbray, "AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis", John Wiley Press, 1998.

[5]. S.R. Chidamber, C.F Kemerer, "A metrics suite for object oriented design**",** IEEE Trans. on Software Eng., vol. 20, no. 6, pp. 476-493, June 1994.

[6]. Santonu Sarkar, Girish Maskeri Rama, and Avinash C. Kak, "API-Based and Information-Theoretic Metrics for Measuring the Quality of Software Modularization," *IEEE Trans. Software Eng.*, vol. 33, no. 1, pp. 14-32, Jan. 2007.

[7]. G. Langelier H. Sahraoui P. Poulin; Visualization based Analysis of Quality for Largescale Software Systems; ASE'05, November 7–11, 2005.

[8]. Ladan Tahvildari, Kostas Kontogiannis, "Improving design quality using meta-pattern transformations: a metric-based approach", Journal of Software Maintenance and Evolution: Research and Practice, John Wiley & Sons Ltd, 2004

[9]. Ralf Reißing, "The Impact of Pattern Use on Design Quality", OOPSLA 2001 Workshop.

[10]. Pattern Inference & Recovery Tool, http://www.cs.ucdavis.edu/~shini/research/pinot2, 2006.

[11]. Metric 1.3.6, A Metric Measuring Tool from Java Source Code, http://www.Sourceforge.net/metric_1.3.6, 2006.