

An Empirical Analysis of Software Systems for Measurement of Design Quality Level Based on Design Patterns

Abstract

In this paper, we propose a new, simple and quantitative approach to specify design level of object oriented software systems. The exploratory analysis method proposed here uses GoF (Gang of Four) design patterns as our assessment criteria. We formulate an empirical study and develop a method to measure software quality. We tested our proposed method on several open source projects and also validate it by making a comparison with current approach. Our approach that addresses design patterns can be an excellent alternative to current systems such as OO metric, software fault proneness, visualization and anti-pattern based approaches. Our approach also can be helpful to practitioners for software quality assurance.

Keywords

Design Pattern, Object Oriented System, Design Quality, OO Metric, Quality Assessment.

I. INTRODUCTION

As software playing a vital role in all sector of our life, and its application growing large in various cases; software managers are focusing on the quality improvement of software. Software system builders increasingly recognize the importance of exploiting design knowledge in the engineering of new systems [1], [2]. So, they are using many methodologies for estimating the quality of software. The idea of design patterns enriched software engineering which has quickly caught the attention of practitioners and researchers and the pattern literature is burgeoning. As design patterns returns a great quality in software, professionals are extensively using design pattern in the design phase of object oriented system. Fig. 1 shows how they use design patterns.

Many models have been proposed to estimate quality of software systems. Many of them based on the prediction of fault proneness of software module [3], [4], [5]. Another approach focuses on detection of anti pattern [6] which is known to be bad coding practices. Some other approaches are based on OO metrics [7], [8] and visualization technique [9]. To the best of our knowledge, in spite of an enormous research and practice have been done about design patterns for several years, there is no standard approach for estimating software design quality by design patterns. The issue remains in the back bench. Our effort tried to encompass this area.

This paper was published in the Procs. of the ICCIT 2007. It is used as an example paper to guide Authors for initial full paper submission for the ICCIT 2008 for blind review

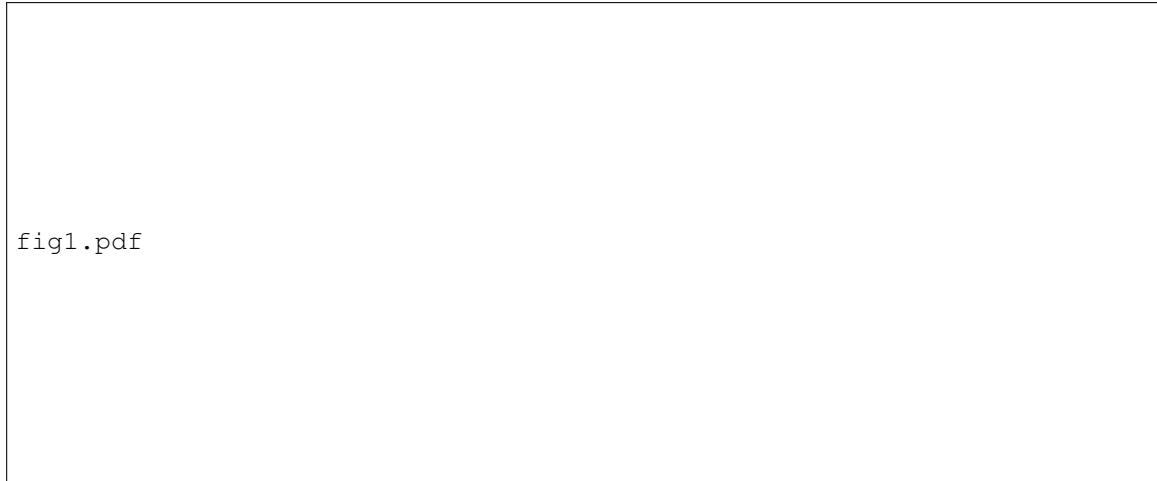


Fig. 1. Abstract software development process [2]

In practice, measurement of software quality is not easy. So, we did an empirical study. The idea of empirically studying phenomena related to software is not new. Many studies have been conducted, especially by the community of software measurement and quality. However, in spite of several quality estimation/prediction models published in the literature [10], [11], concrete applications in industrial contexts are very rare. We validate our proposed formula against OO quality metrics.

The rest of the paper is organized as follows. Section II illustrates about design patterns. Section III describes experimental investigation. Section IV depicts experimental results and comparison with other approaches. Finally section V concludes the paper.

II. 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, 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 problems leading to an advantage for developers [12]. So counting design pattern in large systems can be used to explore their design quality level. Fig. 2 shows how usage of design patterns increasing in OO systems. Here, we measure total patterns in two different versions (version 1.4.3 and version 1.5.6) of java swing package by a tool PINOT (Pattern INference and recOverY Tool)[14].

From Fig. 2, we find that instances of design patterns is higher in the upper version of swing package, i.e. OO community is using more and more design patterns for systems design and implementation as patterns ensures quality design. This is not only true just for swing packages but also for all standard object oriented design. Now, design pattern density is growing high in all OO systems.

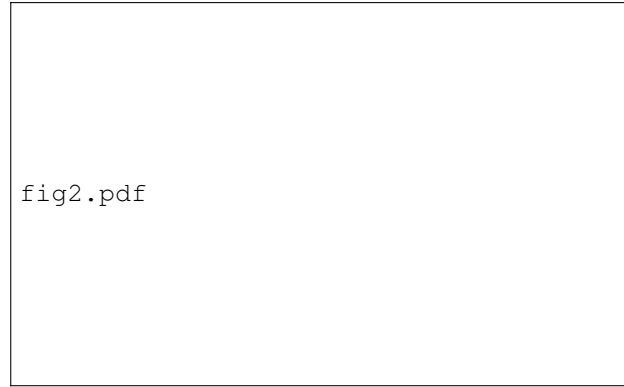


Fig. 2. Comparison of patterns in two different versions of Java Swing Package

III. EMPIRICAL STUDY

A. System Description

When GoF [13] announced 23 design patterns all the people in the object oriented community welcomed it for their great usefulness and effectiveness. In the software industry no question exists about their validity. So, we choose these patterns as our experimental raw material. To demonstrate the idea that design pattern boost software superiority, we break up our experiment into the following steps:

- 1) Firstly, we select some open source projects.
- 2) Then after scrutinizing a lot of tools, we select a tool for mining and counting number of design patterns in several open source projects.
- 3) Two CK metrics [7] and one traditional metric are selected for comparison.
- 4) Patterns are mined by the selected tool.
- 5) Design quality level is measured by our proposed pattern based approach.
- 6) Finally, we compare the obtained results with existing methods.

B. Experimental Setting

1) *Selection of Source Code:* To show that use of design patterns in software design enhance software quality, we select some open source projects. We make a criteria for the collected projects. The three criteria was high, medium and low which indicates design pattern density level. Authors that were primary selected were developers who are recognized, by the object-oriented community. Their code is assumed as good quality code and design pattern's density assumed high and medium there. The search for bad quality code was concentrated on open source and students projects where no or little effort was made in terms of code design. Design pattern density level obviously low there. Good codes are Java swing, JHot Draw, Java util, Fitnessse, JDOM, Deputy and bad codes are students' projects Card Game, Network Protocol, Mobile Game. Table I lists the open source projects' description with design quality level assumption-

2) *Selection of Tool:* There were several tools available for mining design pattern. For example, DP++, SPOOL, Osprey, and Reference extract interclass relationships from C++ source to a database; patterns are then recovered through queries to the database. SOUL is a logic inference system, which has been used to

TABLE I

OPEN SOURCE PROJECTS WITH DIFFERENT DESIGN QUALITY LEVEL

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

recognize patterns (in Java and SmallTalk) based on inter-class-based code idioms and naming conventions. SPQR uses denotational semantics to find patterns on the abstract semantics graphs (ASG) obtained by gcc. FUJABA assumes the existence of a possible pattern and perform a top-down search-to confirm that such a pattern actually exists. But in some cases these finds some wrong pattern. PTIDEJ uses program metrics (such as size, cohesion, and coupling) and a machine learning algorithm to fingerprint roles of a pattern's participating classes. But it is too slow. KT hard-coded its detection algorithms to search for patterns in programs written in SmallTalk. The result was unsuccessful, due to improper message logging mechanism and insufficient test data. We used the tool PINOT [14] which is faster, more accurate, and targets more patterns than existing pattern detection tools. PINOT is a modification of Jikes [15] which is an open source C++ Java compiler. In other tools they fail to identify some patterns though they provide recognition for them. But in this case PINOT is 100% successful. It can identify all the patterns for which it has definition.

TABLE II

OO METRICS SUMMERY

Metric	Object Oriented Feature	Concept	Measurement Method	Interpretation
WMC	Class/Method	Complexity, Usability, Reusability	1) Number of methods implemented within a class 2) Sum of complexity of methods	Larger=> greater potential impact on children through inheritance
CBO	Coupling	Design, Reusability	Number of distinct non-inherited related class inherited	High => poor design, difficult to understand, decrease reuse, increase maintenance
LOC	Size	Complexity	Number of physical lines, statements, and/or comments	Should be small

3) *Selection of OO Metrics*: A healthy amount of research works has been devoted to developing and studying effective software metrics that can be used to measure the quality of an object-oriented design. 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 measure 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).

$$WMC = \sum_{i=1}^n c_i \quad (1)$$

Where c_i is the complexity of a particular method.

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. The larger the number of methods in a class, the greater the potential impact on children since children will inherit all the methods defined in a class. Classes with large numbers of methods are likely to be more application specific, limiting the possibility of reuse. This metric measures usability and reusability. *CBO* is a count of the number of other classes to which a class is coupled. It is measured by counting the number of distinct non-inheritance related class hierarchies on which a class depends. Excessive coupling is detrimental to modular design and prevents reuse. The more independent a class is, the easier it is reuse in another application. The larger the number of couples, the higher the sensitivity to changes in other parts of the design and therefore maintenance is more difficult. Strong coupling complicates a system since a module is harder to understand, change or correct by itself if it is interrelated with other modules. Complexity can be reduced by designing systems with the weakest possible coupling between modules. This improves modularity and promotes encapsulation. *CBO* evaluates design implementation and reusability. We select two OO metric because they measure design complexity, usability and reusability and select traditional metric for standardization of source code. Description of these metrics are given in Table II.

IV. QUALITY ESTIMATION RESULT

A. Patterns per 10 Classes (*P10C*) Method

We mined design patterns by PINOT [14] which runs under Linux environment by the IBM compiler Jikes [15]. It outputs total individual patterns in those projects. For some pattern it symbolizes output by '-' that means it excludes recognition of this pattern. Still there is no tool which can find all the patterns. This tool also outputs name of the classes in which the pattern exist, total number of processed classes, package size some other important information. From that we take into account total number of processed classes. As our formula calculates pattern per 10 classes, total number of processed classes is required to get patterns per 10 classes. We apply pattern mining operation on 9 selected projects. Table III shows total patterns mined in the projects and Fig. 3 shows total patterns after standardization of source code. Fig. 3 makes our guesses true that good codes have high patterns than bad 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. For ensuring quality level we select patterns per 10 classes (*P10C*) as our threshold value.

TABLE III
PATTERN MINING RESULT OF SELECTED PROJECTS

Projects→ ↓Design Patterns	Java Util	JHot Draw	Java Swing	Fitness	JDom	Deputy	Card Game	Network Protocol	Mobile Game
Abstract									
Factory	5	7	66	11	0	0	0	0	0
Factory									
Method	5	7	69	11	0	0	0	0	0
Singleton	0	0	0	0	0	0	0	0	0
Builder	–	–	–	–	–	–	–	–	–
Prototype	–	–	–	–	–	–	–	–	–
Adapter	3	2	29	3	0	0	0	0	0
Bridge	1	5	184	9	1	2	0	0	0
Composite	12	0	19	1	0	3	0	0	0
Decorator	2	1	16	2	2	0	0	0	0
Facade	9	7	174	13	2	6	0	0	0
Flyweight	2	7	75	38	6	5	0	0	0
Proxy	1	2	47	2	0	3	3	3	0
Chain of Responsibility	0	1	15	1	1	0	0	0	0
Command	–	–	–	–	–	–	–	–	–
Interpreter	–	–	–	–	–	–	–	–	–
Iterator	–	–	–	–	–	–	–	–	–
Mediator	15	17	896	32	7	18	0	0	0
Memento	–	–	–	–	–	–	–	–	–
Observer	1	0	70	0	0	9	2	0	0
State	0	11	57	2	0	0	0	0	0
Strategy	3	5	102	7	1	1	0	0	0
Template of Method	0	2	3	1	0	0	0	0	0
Visitor	0	0	3	1	0	0	0	0	0
Total	59	74	1825	134	20	47	5	3	0

The Table IV uses the terms as following meaning:

NoC = No of Processed Class, TP = Total Pattern Found and $P10C$ = Pattern per 10 Classes. Now the formulae for calculating $P10C$ is

$$P10C = \lceil \frac{TP}{NoC} * 10 \rceil \quad (2)$$

A simple example of converting to $P10C$ goes as follows: For java util, $TP = 59$ and $NoC = 101$, then we have,

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

However, we defined a standard of DQL (Design Quality Level) based on this. Table V depicts them. Table IV shows the measured design quality by proposed approach and DQL value assigned based on Table V.

fig3.pdf

Fig. 3. Total patterns after standardization of source code

TABLE IV
DQL OF PROJECTS BASED ON P10C

Projects	NoC	TP	P10C	DQL
Java Util	101	59	6	High
JHot Draw	123	74	6	High
Javax Swing	1485	1825	12	High
Fitness	255	134	5	Medium
JDom	40	20	5	Medium
Deputy	120	47	4	Medium
Card Game	16	5	3	Low
Network Protocol	7	3	4	Medium
Mobile Game	0	0	0	Low

B. Mathematical Model

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

1) *P10C Calculation*: If P_i is a particular GoF Design Patterns, where $i = 1$ to 23, then we define

$$TP = P_1 + P_2 + P_3 + \dots + P_{23} = \sum_{i=1}^{23} P_i \quad (3)$$

and

$$P10C = \frac{TP}{NoC} * 10 \quad (4)$$

2) *DQL Assignment*: As per Table V, we assigned DQL as follows: $DQL = High$ if $P10C$ ranges from 6 to above, $DQL = Medium$ if $P10C$ ranges from 4 to 5, and $DQL = Low$ if $P10C$ ranges from 0 to 3.

C. Result Analysis and Comparison

We measure WMC and CBO from the open source projects by a metric tool metrics 1.3.6 [16]. 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

TABLE V
DIFFERENT METRIC VALUES

<i>P10C</i>	<i>DQL</i>
6 or more	High
4-5	Medium
0-3	Low

different size projects we have to consider a standard volume. We propose here a standard 10K (ten kilo lines) code. Measured values are shown in Table VI.

TABLE VI
DIFFERENT *CK* METRIC VALUES

Projects	<i>LOC</i>	Normal		10K Code	
		<i>CBO</i>	<i>WMC</i>	<i>CBO</i>	<i>WMC</i>
Java Util	20104	25.00	20	12.44	9.95
JHot Draw	21926	15.17	15	6.91	6.84
Javax Swing	134168	32.25	26	2.39	1.93
Fitness	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	28	128.76	600.85

An example of value against 10K is shown below: For Java Util, $LOC = 20104$

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

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

CBO indicates design reusability. High value if *CBO* means poor design. On the other hand *WMC* ensure code complexity, design usability and reusability. Large value of *WMC* have impact on children through inheritance. So, we have to keep it low. Otherwise abstractness of design will go down. We can provide a rank based value of *CBO* and *WMC*. We compare quality level by design pattern based approach and OO metric based approach.

Table VII shows a comparison of design quality level between our pattern based approach, assumed quality level and metrics based quality level. We assumed *DQL* of java util, swing and jhotdraw as high and our approach found so. On the other hand metrics based approach shows their and value better than all others. For Fitness, JDOM, Deputy assumed *DQL* was medium and approach finalizes as medium. Metrics based approach calculate their and value better than students projects but worse than util, swing and jhotdraw. For students projects our assumed *DQL* was low. Pattern based approach encompasses their level as low, medium,

TABLE VII

COMPARISON OF *DQL* WITH *CK* METRIC BASED APPROACH AND DESIGN PATTERN BASED APPROACH

Projects	<i>CBO</i>	<i>WMC</i>	Assumed <i>DQL</i>	<i>DQL</i> Based on Pattern
Java Util	12.44	9.95	High	High
JHot Draw	6.91	6.84	High	High
Javax Swing	2.39	1.93	High	High
Fitness	19.67	10.14	Medium	Medium
JDom	14.33	25.77	Medium	Medium
Deputy	84.35	61.02	Medium	Medium
Card Game	414.00	715.31	Low	Low
Network Prot.	210.26	84.10	Medium	Low
Mobile Game	128.76	600.85	Low	Low

low and metric based approach found their and value worst than all others. We assume students project *DQL* as low but our approach found one as medium. This may occur because some student may use design patterns unknowingly. In this total process firstly, we assume the quality level, then we measure quality level by our formula. Here we found that assumptions are true. Then we make comparison against existing OO metric. Finally after comparison we found that our proposed approach is mostly correct. This experimental result supports a strong base of the proposed approach. It can be a new way of software design quality assurance.

V. CONCLUDING REMARKS

In this paper we have presented an empirical study for OO systems quality analysis and proposed an approach based on design pattern. First, we assume a design level of each open source projects based on their author, and then we perform our experiment. We found that experimental result (design level) is similar to our assumed design level. Finally 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] T. Monroe, A.Kompanek, R. Melton, and D. Garlan, "Architectural Styles, Design Patterns, and Objects", IEEE Software Magazine, January, 1997.
- [2] B.P. Douglass, Doing Hard Time: Developing Real-Time Systems with UML, Objects, Frameworks and Patterns. Addison-Wesley, 1999.

- [3] 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.
- [4] Shi Zhong, M. Khoshgoftaar, and Naeem Seliya, "Expert-Based Software Measurement Data Analysis with Clustering Techniques", *IEEE Intelligent Systems*, Special Issue on Data & Information Cleaning & Preprocessing, 2004.
- [5] 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.
- [6] 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.
- [7] 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.
- [8] 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.
- [9] G. Langelier H. Sahraoui P. Poulin; Visualization based Analysis of Quality for Largescale Software Systems; ASE'05, November 7-11, 2005.
- [10] L.C. Briand and J. Wuest, "Empirical studies of quality models in object-oriented systems". In *Advances in Computers*, 56. Academic Press, 2002.
- [11] N.E. Fenton and M. Neil. "Software metrics: roadmap". In *ICSE - Future of SE Track*, pages 357-370, 2000.
- [12] Detlef Streitferdt, Christian Heller, Ilka Philippow; Searching Design Patterns in Source Code; Proceedings of the 29th Annual International Computer Software and Applications Conference (COMPSAC'05).
- [13] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison Wesley, 1995.
- [14] Pattern Inference & Recovery Tool, <http://www.cs.ucdavis.edu/shini/research/pinot2>, 2006.
- [15] A compiler to convert java files to byte code, <http://ibm.com/developerworks/opensource/jikes>, 2006
- [16] Metric 1.3.6, A Metric Measuring Tool from Java Source Code, <http://www.Sourceforge.net/metric.1.3.6>, 2006.

This paper was published in the Procs. of the ICCIT 2007. It is used as an example paper to guide Authors for initial full paper submission for the ICCIT 2008 for blind review