



Criteria Based Evaluation Pattern for Software Design Model

Rashmi Rekha Sahu*School of Computer Engineering
KIIT University,
Bhubaneswar, India**Abhishek Ray**School of Computer Engineering
KIIT University,
Bhubaneswar, India**Durga Prasad Mohapatra**Department of Computer Science Engineering
National Institute of Technology,
Rourkela, India

Abstract— *Software design evaluation plays an important role in software development process, to generate software product with high levels of productivity and efficiency. To achieve high level of qualitative software product, implementation of pattern can create a domain specific framework to provide consistency throughout a software solution. This paper proposed an evaluation pattern called Criteria Based Evaluation Pattern (CBEP) for measuring the Object-Oriented Design (OOD) model. First, we have identified some criteria and parameters by considering the design model. Then different metrics are proposed for each criteria and these metrics are used for measuring the design model.*

Keywords— *Pattern, OOD, Software design model, Metrics*

I. INTRODUCTION

The goal of software development is to develop qualitative software products with high levels of productivity and efficiency. To ensure the productivity and efficiency of software product, software design phase plays a vital role in Software Development Life Cycle (SDLC) [6]. To achieve this, the use of patterns for evaluation of the software design can stand out as one of the most acceptable mechanism. A pattern is the abstraction from a concrete form which keeps recurring in specific non-arbitrary contexts. Each pattern consists of three part rule, which expresses a relation between a certain context, a problem and a solution. There are varieties of pattern present for software development process i.e. architectural pattern, process pattern, design pattern etc.

Architectural patterns are used for solving the design problem in the field of software architectures. Process pattern is a mechanism which represents an abstraction to commonly recurring activities in software development [3]. Design pattern addresses a recurring design problem for software system by describing the problem, solution, when to apply the solution and its consequences. The solution is a general arrangement of objects and classes that solve the problem and can be implemented for a particular context by customizing it in a proper format [1]. The above existing patterns are used for addressing the design problems in their respective domain. From empirical studies we found that these patterns are not used for evaluation of design models. As it expresses a fundamental structural organization or schema for software system, which is used only for improving the software design model. Though software design evaluation is an aid of software development and aims at the determination for degree of desired qualities of a finished system, there is a necessity of developing a general pattern which can be used for measuring design model.

In this paper we have proposed a general pattern called Criteria Based Evaluation Pattern (CBEP). This pattern consists of some basic criteria defined in ISO-9126 [9]. These criteria means the measurable part of quality attributes of design or evaluation. Then different metrics are proposed for each selected criteria and these metrics are used for the evaluation of design model.

The rest of the paper is organized as follows: Section II describes the related work. The proposed approach is elaborated in Section III. In Section IV different metrics are proposed and Section V shows the working of the proposed framework using a case study. Section VI concludes the paper along with the future work.

II. RELATED WORK

In this section we have performed a survey on software design evaluation and software pattern for software development process. Our work focus on finding the existing metrics which are used for design evaluation. The various information related to these existing metrics has been discussed below.

Genero et al. [4] has analyzed a set of existing object-oriented metrics. They have presented some relevant measurable elements for evaluating the class complexity at the initial phase of the development life cycle. The author has also proposed some new metrics related to relationships for both class and packages using same criteria like association, aggregation and dependency. In this paper the new metrics are only defined but are not empirically validate.

Abreu et al. [2] the author has adopted a suite of metrics to measure the object-oriented design (OOD) characteristics. These metrics helped in evaluation of software quality characteristics such as defect density and rework by means of experimental validation.

Thirugnanan et al. [8] quality metric tool has been developed. This metric tool determines a number of design metrics such as project wide metrics, module wide metrics, C.K. metrics, object-oriented metrics and quality attributes.

Huston [5] has considered the design metric along with design pattern for determination of their compatibility in terms of their application for improvement of design quality. He has used some metrics under the design patterns and has compared them by using proposed pattern with non-pattern design. Some issues like enhancement of loose coupling among a group of classes, decoupling of an abstraction from its implementation have been handled by proposing suitable design pattern and also evaluated by using metrics.

III. PROPOSED APPROACH

This section presents our proposed framework. The main goal of our proposed work is to develop a common criteria based evaluation pattern based on design model for software development process, is represented as a general framework as shown in Fig. 1. In our proposed approach we have considered an object-oriented design model of a system. In the next step, for the purpose of design evaluation a set of quality criteria is to be identified. Then a minimum set of criteria are selected through which a system design model can be evaluated. After selection the criteria are evaluated using the proposed metrics and finally a pattern is proposed based on these criteria called criteria based evaluation pattern. As ISO/IEC 9126 standard describes desirable quality characteristics that are needed for the evaluation of software, in this context we have selected four minimum quality characteristics i.e. reliability, efficiency, size, maintainability. These four quality characteristics are used for evaluation of software design model. These characteristics can be defined by using different criteria. The criteria are measured by means of software design attributes (object-oriented attributes) known as metrics and these metrics are used to check the efficiency of each criteria.

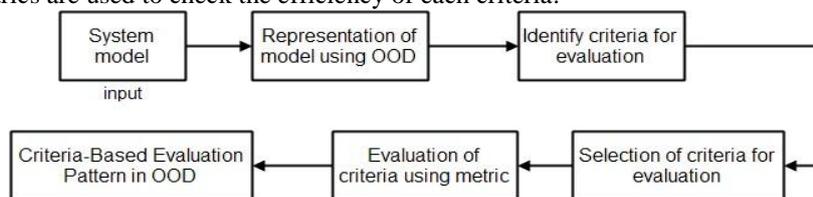


Fig. 1 General framework for criteria based evaluation pattern

IV. PROPOSED METRIC FOR IDENTIFIED CRITERIA

A. Clarity

Object-oriented design model should be precise, clearly defined and easily understood to the system developer. Objects, methods and relationship between objects should be designed explicitly and unambiguously, so that the developer could able to understand the design easily. To measure the above mentioned issues object-oriented model is represented in a tree structure to convince all the characteristics of clarity. A sample tree structure is shown in Fig. 2 which represents the relationship and hierarchy between different methods and their respective class. Each method can further be depicted through the functions that are present in that particular class. Similarly all the classes present in the system can be represented in tree structure.

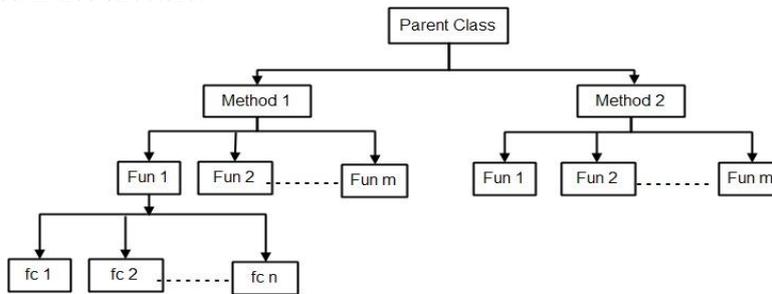


Fig. 2 Tree Representation of Software design model

B. Complexity

The design model should be represented in a better way to minimize the complexity. There are two types of complexity i.e. space complexity and time complexity [7]. Space complexity is the total memory space taken by total number of classes. Time complexity is a measure of the total amount of time required to execute the whole system. If the design model is providing lower time complexity then it can be considered as one of the most desirable design model. Here we have considered time complexity as a measuring parameter to determine the total time required to execute a system model. This measuring parameter can also be called as a metric.

Let q be the number of methods present in a system design model and m be the number of functions available in each method. Considering each function contains n number of child functions, the time complexity of system can be computed using Eq. 1.

$$T_m = \sum_{k=1}^q (T_f)_k \quad (1)$$

where T_m = Time complexity of the system
 T_f = Time complexity of function.

T_f is calculated using Eq. 2:

$$T_f = \sum_{j=1}^m (T_{fc})_j \quad (2)$$

where T_{fc} = Time complexity of the child function

T_{fc} can be defined in Eq. 3:

$$T_{fc} = \sum_{i=1}^n (T_{fc})_i \quad (3)$$

where $(T_{fc})_i$ = Time complexity of sub-child function

C. Coverage

Object-oriented model should cover the whole generic life cycle. Here we have considered the criteria coverage which measures to what extent a method/function of a class cover the whole system. This criteria can be measured through the code metrics. Code metric is a set of software measures that provide developers better insight into the code they are developing. By taking advantage of code metrics developers can understand which methods and/or functions should be reworked. This coverage criteria can be measured through the metric Lines of code (LOC). The metric LOC indicates the approximate number of lines in the code. The count is based on the number of lines in the source code file. A very high count might indicate that a type or method is trying to do too much work and should be split up. It also indicates that the type or method might be hard to maintain. If a system model contains n number of classes i.e. C_1, C_2, \dots, C_n then total lines of code (LOC) of the system can be computed using Eq. 4.

$$\text{Total LOC} = \text{LOC of } C_1 + \text{LOC of } C_2 + \dots + \text{LOC of } C_n \quad (4)$$

Where $\text{LOC of } C_i = \text{LOC of each individual class } C_i$ and $i = 1, 2, \dots, n$.

The percentage coverage can be find out as follows:

$$\% \text{ coverage of each class} = \frac{\text{Total LOC of class}}{\text{Total LOC of the system}} \quad (5)$$

D. Extensibility

In software engineering, extensibility is defined as a systematic measure of the ability to extend a system model and the level of effort required to implement the extension. It measures the extent to which the system design model can be enriched with additional concepts. The criteria extension is associated with flexibility and adaptability. To measure extensibility, we have only considered the flexibility aspect as our approach is limited to design model. Flexibility is parameter which refers to the ability that respond to the changes occur due to the addition of new fragments. It can measured through the degree of the encapsulation, coupling, polymorphism, composition. So in OOD model extensibility can be represent as follows:

$$\text{Extensibility} = E - C + P \quad (6)$$

where E = Encapsulation

C = Coupling

P = Polymorphism

Encapsulation is a property which binds the data and function into single unit. In object-oriented design data encapsulation is the most striking feature of class. The degree of encapsulation depends on the metrics percent public data (% public data) and percent access to public data (% access to public data).

$$\text{Encapsulation} = 1 - (\% \text{public data} + \% \text{access to public data}) \quad (7)$$

Where percent public data is defined as percentage of public and protected data within a class and percent access to public data is the number of access to public and protected data. If percent public data and Percent access to public data is less than 1 (Where 1 represents all private, public and protected data of the whole system) then the degree of encapsulation will be more which can be considered as a positive degree. Coupling between to classes is a measure of how much they depend on each other (degree of dependency). The degree of coupling is represented in Table I.

Table I: Table for degree of Coupling

Relationship between classes	Degree of coupling factor
one to one	low
one to many	low
many to one	high
many to many	high

Polymorphism is the ability to manipulate objects of distinct classes using only knowledge of their common properties without regard for their exact class. The degree of Polymorphism depends on Percentage of Overloaded Methods (POM) present in the system. If POM is high then extensibility of the system design model is high.

Mathematically,
$$\text{POM} = \frac{\text{Total number of overloaded methods}}{\text{Total number of methods present in the system}} \times 100 \quad (8)$$

V. CASE STUDY : HOSPITAL MANAGEMENT SYSTEM

This section illustrates the implementation of our proposed framework. By using the proposed framework the quality of the system model is measured through the criteria based evaluation pattern. This pattern helps in evaluating the software design model. We have considered the class diagram of Hospital Management System (HMS) as a system design model. The class diagram describes the overall structure of HMS by showing the systems classes, attributes, methods and relationships among the classes. It considers eight classes such as Login, MDIMain, Worddetails, Outpatient, Doctors, Employees, Services and Applications. The system has the scope of adding new employees, doctors admitting outpatients, fixing appointments etc. This project also displays the services provided to outpatient.

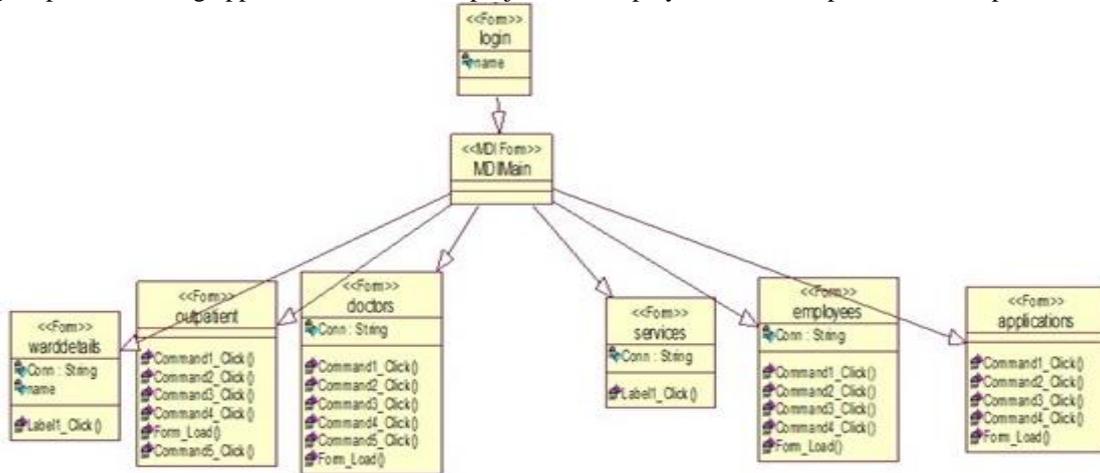


Fig. 3. Class Diagram of HMS

A. Clarity

The class diagram of HMS in Fig 3 is represented into tree structure in Fig 4. Tree representation shows the class hierarchy of HMS which makes the system design model more simple, understandable and more clear. In the class diagram of HMS, MDIMain is considered as parent class because it provides base screen for all management program actions present in the management program. It contains menu bar which list the manipulation forms for management. By using this tree representation the complexity of the system has been calculated.

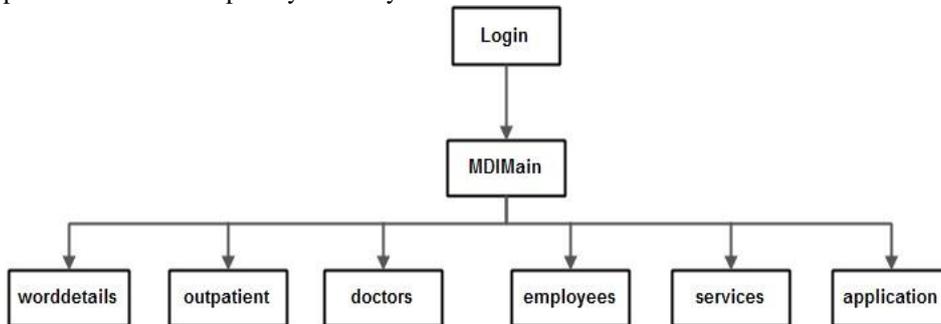


Fig. 4. Tree representation of Hospital Management System (HMS)

B. Complexity

In the case study of HMS model each function that are defined in the class has been shown in Table II then the time complexity of the complete system is calculated. In Table II the time complexity of each function is shown and accordingly the time complexity of each class is calculated. Similarly time complexity of the complete system is calculated as shown in Eq.9.

Table II: Time complexity for HMS

Class Name	Function Name	TC of each function T_{fc}	Total TC of each class $T_f = \sum_{j=1}^m (T_{fc})_j$
worddetails	Label1_click()	n	n
outpatient	Command1_click()	1	4n+2
	Command2_click()	n	
	Command3_click()	n	
	Command4_click()	n	
	Command5_click()	1	

	Form_load()	n	
doctors	Command1_click()	1	5n+1
	Command2_click()	n	
	Command3_click()	n	
	Command4_click()	n	
	Command5_click()	n	
	Form_load()	n	
services	Label1_click()	n	n
employees	Command1_click()	1	4n+1
	Command2_click()	n	
	Command3_click()	n	
	Command4_click()	n	
	Form_load()	n	
applications	Command1_click()	1	2n+3
	Command2_click()	1	
	Command3_click()	n	
	Command4_click()	n	
	Form_load()	1	

$$T_m = \sum_{k=1}^q (T_f)_k = (T_f)_1 + (T_f)_2 + (T_f)_3 + (T_f)_4 + (T_f)_5 + (T_f)_6 = O(n) \quad (9)$$

C. Coverage

The above mentioned Fig 3 contains eight classes and each class contains a number of methods. The size or coverage of each methods are shown in the Table III and by using this percentage coverage has been calculated.

Table III: Calculation for percentage coverage of HMS

Sl. No.	Class Name	Size (LOC)	Percentage Coverage
1	login	12	4.82
2	MDIMain	28	11.24
3	worddetails	15	6.02
4	doctors	73	29.32
5	outpatient	42	16.86
6	employees	45	18.07
7	Services	17	6.83
8	applications	17	6.83
		Total LOC=249	

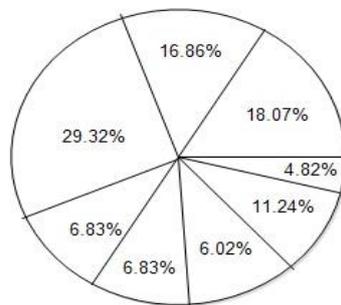


Fig. 5. Percentage coverage of HMS

E. Extensibility

According to the Eq. 6 extensibility criteria depends on the degree of encapsulation, coupling, polymorphism.

- Degree of encapsulation depends on percentage of public data and percentage of access to public data as shown in Eq.7.

In HMS design model word details class doesn't contain any public data it contains only private data. Therefore %public data and %access to public data is zero. So degree of polymorphism is 1.

$$\text{i.e. } E = 1 - (0 + 0) = 1. \quad (10)$$

Similarly the degree of encapsulation can be obtained for other classes.

- According to Eq.8 degree of polymorphism depends on total no. of overloaded methods and the total no. of methods present in the system.

Outpatient and Employees are two classes present in HMS. Outpatient contains 6 methods and Employees contains 5 methods. The 5 methods of employee class are overloaded with the methods present in Outpatient class. Similarly the methods present in these classes can also overloaded with method present in other classes.

Therefore degree of polymorphism is:

$$P = \frac{7}{24} \times 100 = 29.16\% \quad (11)$$

- Degree calculation of coupling:
Table IV shows degree calculation of coupling for HMS according to the relationship and the type of coupling between the classes. These relationship and classification of coupling is arranged from low to high degree of coupling. Data coupling has lowest and content has highest degree of coupling for all type of relationships between classes i.e. one to one, one to many, many to one and many to many.

Table IV: Degree calculation of coupling for HMS

Relationship between classes	Data coupling	Stamp coupling	Control coupling	Common coupling	Content coupling
one to one	2	3	4	5	7
one to many	3	5	6	8	9
many to one	4	6	7	7	9
many to many	5	7	8	8	10

VI. CONCLUSIONS

In this paper we have proposed criteria based evaluation pattern for software design model. For developing the evaluation pattern a set of quality criteria are identified and evaluated using proposed metrics. The working of this pattern has been tested for class diagram, activity diagram and sequence diagram. This pattern can further be used for service-oriented system, real-time system or aspect-oriented system etc.

REFERENCES

- [1] "what is a software pattern." wiki.gxtechnical.com/commwiki/servlet/hwiki?what+is+a+software+pattern.
- [2] F. Brito e. Abreu and W. Melo, *Evaluating the impact of object-oriented design on software quality*, 3rd International Software Metrics Symposium (METRICS96), IEEE, Berlin, Germany, March, 1996.
- [3] M. Fahmideh, P. Jamshidi, and F. Shams. *A procedure for extracting software development process patterns*, UKSim Fourth European Modelling Symposium on Computer Modelling and Simulation, pp 75–83, 2010.
- [4] M. Genero, M. Piattini, and C. Calero. *Early measures for uml class diagrams*, *University of Castilla –La Mancha*, vol. 6, issue. 4, 2000.
- [5] B. Huston, The effects of design pattern application on metric scores, 3rd The journal of systems and software, vol. 58, pp 261– 269, September, 2001.
- [6] R. Mall. *Fundamental of Software Engineering*, PHI Learning Private Limited, New Delhi, 2009.
- [7] F. T. Sheldon, K. M. Daley and H.Chung, *Measuring the complexity of class diagrams in reverse engineering*, *The Journal of Software Maintenance and Evolution: Research and Practice*, pp 1–14, July, 2005.
- [8] M. Thirugnanam and Swathi.J.N, *Quality metrics tool for object oriented programming*, *International Journal of Computer Theory and Engineering*, vol. 2, issue. 5, pp 1793–8201, October, 2010.
- [9] D. Wallace and L. Reeker, *SWEBOK*, IEEE Computer Society, Los Alamitos, California, 2004.