ISSN:2319-7900

A FRAMEWORK TO MEASURE THE QUALITY OF SOFTWARE THROUGH EVALUATION USING OOAD METRICS

Mr.S.Pasupathy, Associate Professor; Dr.R.Bhavani, Professor;
Dept. Of CSE, Annamalai University, Tamilnadu, India;
(e-mail: [1]pasuannamalai@gmail.com,[2]shahana_1992@yahoo.co.in)

Abstract

With the tremendous increase in the growth of computer, everything has been computerized. As technology develops, everything has been modernized. In human life, computer and its usage occupies major part in our day-to-day activities. Most of our daily activities depend on computer. For example, E-Ticket, E-Shopping, E-Learning and so on.

To make the process computerized, everything has been programmed on computer. Based on the purpose, the program can be categorized. To make the program simpler and readable, OOAD can be used to develop the program based on Object-Oriented. The program developed on OOAD can be modular and functional.

Each program must be of best quality. The quality of the program can be defined by measuring the error rate of the program on compiling. The program can be evaluated in number of phases by different level of programmers. After completing all the phases, the error rate in the program can be calculated and evaluate the percentage of marks. Based on the percentage, the quality of software can be measured. This can be proposed in this paper with efficient methodology.

Introduction

Computer is an electronic machine but it occupies major role in the human life. Computer reduces the manual effort and performs the function much faster. So that, it becomes more popular and essential in our life. Since, computer is a machine and so it can be functioned only with the help of user-defined program. Based upon the purpose, the program can be defined in number of ways. The program may be 10 lines or 100 lines or any more lines as per the purpose of the program and the user requirement. As the number of lines of code increases, the readability of the program becomes difficult. The good programmer is the one who develops the program with less number of lines of code.

To develop the program, many technologies and languages are there. One such technology is "**Object Oriented Analysis and Design (OOAD**)". OOAD is a technique that consists of lot of metrics to validate and measure the quality of the software.

OOAD is a software engineering approach that models a system as a group of interacting objects. An object-oriented system is composed of objects. The behavior of the system results from the collaboration of those objects. Collaboration between objects involves those sending messages to each other. OOAD is comprised of two parts:

- Object oriented analysis
- Object oriented design

Models of different types can be created to reflect the static structure, dynamic behavior, and run-time deployment of the collaborating objects of a system.

During the **object-oriented analysis** (OOA) phase object-modeling techniques are used to analyze the functional requirements for a system and create models which reflect the logical design of the system. OOA focuses on studying and understanding the problem first, initially ignoring the concerns of an actual implementation. Many terms are used to describe the key real-world concepts of an application.

- ➢ Problem Domain
- ➢ Application Domain
- ➢ Business Objects
- ➢ Domain Objects
- ➢ Problem Essence
- ➢ Key Classes

During the **object-oriented design** (OOD) phase of the system, models are elaborated upon to include implementation specific details that show how the physical design of the system will come together. It is viewed as an extension of analysis, more so than a distinct activity. OOD adds the detailed design for each class:

- Names, data types, and access specifiers for attributes.
- Names, return types, and parameter lists for all methods.
- Associations and collaborations with other classes.

OOA focuses on what the system does (its static structure and behavior), OOD on how the system does it (it's run-time implementation).

The creation of program consists of following process:

- ✓ Writing the program
- ✓ Compiling the program
- ✓ Executing the program.

The program written on any language must be compiled using the compiler to check for errors in the program. The compiler reads the program, line by line and then analyzes the errors. The error may be of different kinds such as syntax error, data type mismatch, missing colon, semicolon and so on. When the compiler finds the error, it displays the error to the user to clear it. Only after correcting all the errors, the final step of execution takes places.

In this paper, we have to measure the quality of program. The quality of the program can be measured in various ways. But we measure the quality of program by calculating the error rate. In previous research work, the method to calculate the error rate was described by analyzing the type of error occurs. In this paper, we categorize the error and based upon the error rate, the percentage of mark has to be evaluated. Thus, based on the percentage of marks, the quality of program can be measured. Thus the proposed strategy provides efficient methodology to implement the object-oriented metrics.

Previous Research

In paper [1], Deepak et al described that Software metrics are required to measure quality in terms of software performance and reliability related characteristics like dependencies, coupling and cohesion etc. It provides a way to measure the progress of code during development and having direct relationship with cost and time incurred in the software design and development at their later stages. These major issues must be checked and informed early in the development stage, so that reliability of any software product could be ensured for any large and complex software project. Object oriented software metrics directly focuses on the issues like complexity, reliability and robustness of the software developed using object oriented design methodologies. It reflects the time, cost and effort that would be incurred in development at later stage. While the software in its development stage, it was desirable that the complexity levels at every stage should be minimized to make the end product more reliable and manageable. Object oriented metrics provides all parameters through which one can estimate the complexities and quality related issues of any software at their early stages of development. In the paper, authors have studied three object oriented metrics namely MOOD Metrics, CK Metrics, and QMOOD Metrics and given a case study to show, how these metrics are useful in determining the quality of any software designed by using object oriented paradigm.

In paper [2], Henderson described that Object oriented approach was capable of classifying the problem in terms of objects and provide many paybacks like reliability, reusability, decomposition of problem into easily understood object and aiding of future modifications.

In paper [3][4][5], Briand et al stated that Object-Oriented Metrics are useless if they are not mapped to software quality parameters. Many number of quality models are proposed to map parameters of the Object Oriented software like Extensibility, Reusability, efforts, manageability and cost. To know more about the internal structure of the product one should know more about the interdependencies of parameters of metrics and Software quality parameters.

In paper [6][7], National Institute of Standards and Technology and Sutherland reported that Incorrect and buggy behavior in deployed software costs up to $70 billion each year in the US. Thus debugging, testing, maintaining, optimizing, refactoring, and documenting software, while time-consuming, remain critically important. Such maintenance was reported to consume up to 90% of the total cost of software projects [8]. A key maintenance concern was incomplete documentation [9]: up to 60% of maintenance time was spent studying existing software (e.g., [10]).Human processes and especially tool support for finding and fixing errors in deployed software often require formal *specifications* of correct program behavior (e.g., [11]); it was difficult to repair a coding error without a clear notion of what "correct" program behavior entails. Unfortunately, while low-level program annotations are becoming more and more prevalent [12], comprehensive formal specifications remain rare. Many large, preexisting software projects are not yet formally specified

ISSN:2319-7900

[12]. Formal program specifications are difficult for humans to construct [13], and incorrect specifications are difficult for humans to debug and modify [14].

In paper [15], Halstead reported that a full survey of software quality metrics was outside the scope of the article; instead, they highlight several notable approaches. Halstead *et al.* proposed *Software Science* (which did not prove accurate in practice [16]), to provide easily measurable, universal source code attributes.

In paper [17], Gabel et al described that as these large specifications are imprecise and difficult to debug, this article focuses on a second class of techniques that produce a larger set of smaller and more precise candidate specifications that may be easier to evaluate for correctness. These specifications typically take the form of two-state finite state machines that describe temporal properties, e.g. "if event *a* happens during program execution, event *b* must eventually happen during that execution." Two state specifications are limited in their expressive power; comprehensive API specifications cannot always be expressed as a collection of smaller machines.

In paper [18], More recently, Nagappan and Ball analyzed the relationship between software dependences, code churn (roughly, the amount that code has been modified as measured by source control logs), and post-release failures in the Windows Server 2003 operating system.

In paper [19], Graves et al described that they show that *relative* code churn, or the amount of churn in one module as compared to a dependent module, is more predictive of errors than *absolute* churn (which we use here). This suggests that more sophisticated measures of churn might be more predictive in our model. They similarly attempt to predict errors in code by mining source control histories.

In paper [20], Albrecht described that Function Point Analysis (FPA) estimates value delivered to a customer, who can help approximate, for example, an application's budget, the productivity of a software team, the software size or complexity, or amount of testing necessary.

Research work

The aim of the paper is to analyze the program to measure the quality of the program by evaluating the percentage of error rate occurred in the developed program. This can be achieved by using OOAD metrics. OOAD metrics are of numerous to measure the software in different ways.

As described earlier, the program written in OOAD, has been compiled to check for errors. When the error has been identified, it is to be categorized based on the type of errors and it can be saved till the end of the program. Finally, the error rate has been calculated and displayed to the user.

In this paper, the methodology has been extended to evaluate the percentage of error rate and based on the resultant percentage; the quality of software has to be measured. The summary of the research work is discussed below:

In a company, there may be different levels of programmers, who can develop the program, compiling the program and so on. The number of programmers and the level of programmers can be varied depend upon the company standard. Based upon the company standard, we define several parameters to evaluate the program.

One such parameter is to categorize the programmer into several levels of programmers, such as ASE (Associate Software Engineer), SSE (Senior Software Engineer), Program Analyst, System Engineer, Junior Programmer.

These levels of programmers can be assigned with certain roles and specified rules. In addition to that, we have to specify the maximum error rate that can be allowable for that level of programmer. This error rate can be allocated based upon their level.

This can be defined below:

ASE	a%
SSE	b%
Program Analyst	c%
System Engineer	d%
Junior Programmer	e%

Once the program has been developed, it undergoes the process of compilation. Upon compiling, the details such as who compile the program, what type of error occurs in that program has been notified and tabulated as follows:

Programmer Level	Type of Error	Line Number	Total Errors
ASE	Syntax Error	2	2
ASE	Data-type mismatch	7	
SSE	Run-time error	3	3
SSE	Missing Colon/Semicolon	4	
SSE	Other error	5	
…	…	…	…

For each kind of error, the line number where the error occurred in the program can also be noted. In addition to the specified parameters, one more parameter is to be defined in this paper to award the score or mark for the error occurred. For each type of error, the mark has to be specified and it can be calculated for the identified error occurred in the program to measure the quality of the program.

Syntax Error	2
Run-time Error	5
Data-type mismatch	3
Missing Colon / Semicolon	1
Other Error	7

The next step of identifying the type of error is to calculate the marks to be awarded. Finally the percentage of marks can be calculated by dividing the number of lines of code by the calculated total marks.

$$\% \text{ of Marks} = \frac{LOC}{Total}$$

Where,
LOC – Lines of Code
Total - $\sum_{i=1}^{n}$ (No. of Mistakes * Marks Awarded)

Finally, the quality of program can be evaluated by analysing the level of programmer and then by compare the maximum percentage of error occurred in the program.

Consider, the maximum percentage of error allowed to the programmer of level p1 be n%, the quality of program can be calculated by comparing the percentage of marks with programmer percentage. Suppose the percentage of marks calculated be X, the quality of program can be measured as follows:

If $X < n$, quality = Good
If $X > n$, quality = Bad

Based upon the quality measured, we categorize the program whether to be useful or not. These are all specified in the XML file which is given below:

```
<XML version = 1.0>
<!—Error Definition →
<level1>
        <Programmer> ASE </Programmer>
        <Error_Rate>    a% </Error_Rate>
</level1>
<level2>
        <Programmer> SSE </Programmer>
        <Error_Rate>    b% </Error_Rate>
</level2>
<level3>
        <Programmer> Program Analyst </Programmer>
        <Error_Rate>    c% </Error_Rate>
</level3>
<level4>
        <Programmer> System Engineer </Programmer>
        <Error_Rate>    d% </Error_Rate>
</level4>
<level5>
        <Programmer> Junior Programmer </Programmer>
        <Error_Rate>    e% </Error_Rate>
</level5>
</XML>
```

```
<XML version = 1.0>
<!—Program Evaluation →
<Error1>
        <Type>Syntax Error </Type>
        <Mark> 2 </Mark>
</Error1>
<Error2>
        <Type>Run-Time Error </Type>
        <Mark> 5 </Mark>
</Error2>
<Error3>
        <Type>Data-type Mismatch </Type>
        <Mark> 3 </Mark>
</Error3>
<Error4>
        <Type>Missing Colon / Semicolon </Type>
        <Mark> 1 </Mark>
</Error4>
<Error5>
        <Type> Other Error </Type>
        <Mark> 7 </Mark>
</Error5>
</XML>
```

The outline of the methodology is given below: The quality of the program can be evaluated by considering the following criteria:

- Number of Lines of Code
- Level of Programmers
- Defined Error Rate
- Calculated Error Rate
- Quality of Program

Thus the proposed methodology provides the efficient way to measure the quality of program. The proposed method also contains the following algorithm which works based on the given conditions:

Thus the proposed methodology provides the efficient way to measure the quality of program. The proposed method also contains the following algorithm which works based on the given conditions:

Algorithm

```
Start
Categorize the level of programmers, $P_i$
Define the error rate for each level of programmers, Rate ($P_i$)
Define the Marks for the error, Mark (Error)
Analyze the program, P
Identify the errors, err
Calculate the Error Rate
Calculate the LOC, loc
Rate of Error = LOC / Error Rate
Identify the level of Programmer, $P_x$
Identify the allowable error rate, rate ($P_x$)
If rate ($P_x$) > Rate of Error then
        Quality = "Good"
Else
        Quality = "Bad"
End if
End
```

Experimental Results

Consider the software organization have following maximum acceptable error rate for each category of software developers

ASE	1%
SSE	2%
Program Analyst	6%
System Engineer	8%
Junior Programmer	10%

Table for marks awarded for each type of error

Syntax Error	2
Run-time Error	5
Data-type mismatch	3
Missing Colon / Semicolon	1
Other Error	7

In our Experiment we consider SSE(Senior Software Engineer) develops the following java program with error,

```
/*senior software engineer(SSE)*/
import java.io.*;
class add
{
        pulic static void main(String args[])
        {
                int a,b
                float c;
                a=10;
                b=0;
                c=a/b;
                System.out.println("add="+c);
        }
}
```

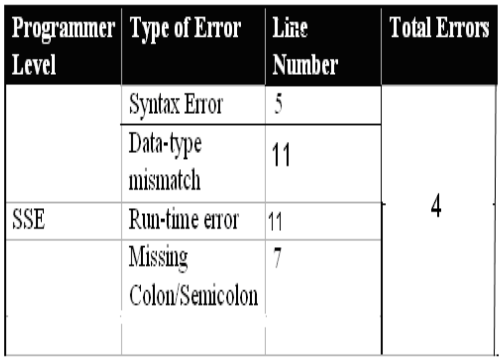
Mathematics

Math typesetting can be done by Equation Editor, or by any other system that produces clear math types. Symbols and shorter expressions can be placed within the text, e.g., $\lambda_i \to 0$ and $P_s \le P_r$. More complex expressions should be placed in a new line in display style:

$$P(k) = \binom{m}{k} P_1^k (1 - P_1)^{m-k} \qquad (1)$$

All equations should be numbered and placed in the center of the column (using Tab key). Equation number should be flushed to the end of the column (using Tab key). They should be referenced like Equation (1). Unless it is absolutely necessary, equation numbers should not have parts to them. E.g., instead of using Equation 1(a) and Equation 1(b), please number them as Equation (1) and Equation (2)

The following table is the list of errors obtained by running the above java code.



Programmer Level	Type of Error	Line Number	Total Errors
	Syntax Error	5	4
	Data-type mismatch	11	
SSE	Run-time error	11	
	Missing Colon/Semicolon	7	

The above program has:

Total no of lines=14

Total Marks for error=11

% of Marks =14/11 =1.2%

So the % of marks for SSE does not reach the maximum acceptable error rate of 2%, so SSE can continue their work to complete the project.

Our methodology is very efficient that is used to evaluate the quality of the software. To test the efficiency of the methodology, various experimental setups are constructed and the result is analyzed.

Our proposed methodology has been experimented by taking a software company to measure the quality of program developed by different level of programmers. The program can be developed, and then it undergoes the compilation process. Upon identifying the error, the error rate has been calculated and finally the quality of program has been measured successfully.

Thus our proposed methodology provides better experimental results and it is very much useful for all kinds of users to analysis the quality of the software.

Conclusion

The aim of the paper to measure the quality of program has been successfully implemented in the proposed methodology and various experimental setup has been undertaken to evaluate the methodology.

Thus, the proposed method of this paper performs better to measure the quality of software by calculating the error rate occurred in the program. Thus our research work completed successfully with efficient methodology proposed in this work.

References

[1] Deepak Arora, Pooja Khanna and Alpika Tripathi, Shipra Sharma and Sanchika Shukla, Faculty of Engineering,Department of Computer Science, Amity University, “Software Quality Estimation through Object Oriented Design Metrics”.

[2] J. Alghamdi, R. Rufai, and S. Khan. Oometer: A software quality assurance tool. Software Maintenance and Reengineering, 2009. CSMR 2009. 9th European Conference on, pages 190{191, 21-23}, March 2010.

[3] L.C.Briand, J.Wuest, J.Daly and Porter V., “Exploring the Relationships Between Design Measures and Software Quality In Object Oriented Systems”, Journal of Systems and Software, 51, 2000.

[4] 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, 2002.

[5] P.Coad and E.Yourdon, “Object Oriented Analysis”, Yourdon Press, 1990.

[6] National Institute of Standards and Technology, “The economic impacts of inadequate infrastructure for software testing,” Tech. Rep. 02-3, May 2002.

[7] J. Sutherland, “Business objects in corporate information systems,” *ACM Comput. Surv.*, vol. 27, no. 2, pp. 274–276, 1995.

[8] R. C. Seacord, D. Plakosh, and G. A. Lewis, *Modernizing Legacy Systems: Software Technologies, Engineering Process and Business Practices*, 2003.

[9] S. C. B. de Souza, N. Anquetil, and K. M. de Oliveira, “A study of the documentation essential to software maintenance,” in *SIGDOC*, 2005, pp. 68–75.

[10] S. L. Pfleeger, *Software Engineering: Theory and Practice*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2001.

[11] D. Malayeri and J. Aldrich, “Practical exception specifications.” in *Advanced Topics in Exception Handling Techniques*, 2006, pp. 200– 220.

[12] M. Das, “Formal specifications on industrial-strength code — from myth to reality,” in *Computer-Aided Verification*, 2006, p. 1.

[13] H. Chen, D. Wagner, and D. Dean, “Setuid demystified,” in *USENIX Security Symposium*, 2002, pp. 171–190.

[14] G. Ammons, D. Mandelin, R. Bod´ık, and J. R. Larus, “Debugging temporal specifications with concept analysis,” in *Programming Language Design and Implementation*, 2003, pp. 182–195.

ISSN:2319-7900

[15] L. Briand, S. Morasca, V. Basili, Property-Based Software Engineering Measurement, IEEE Trans. Software Eng. 22(1), 2000, pp. 68-85.

[16] P. G. Hamer and G. D. Frewin, "M.H. Halstead's Software Science - a critical examination," in *ICSE*, 1982, pp. 197–206.

[17] M. Gabel and Z. Su, "Symbolic mining of temporal specifications," in *ICSE*, 2008, pp. 51–60.

[18] N. Nagappan and T. Ball, "Using software dependencies and churn metrics to predict field failures: An empirical case study," in *ESEM*, 2007, pp. 364–373.

[19] T. L. Graves, A. F. Karr, J. S. Marron, and H. Siy, "Predicting fault incidence using software change history," *IEEE Trans. Softw. Eng.*, vol. 26, no. 7, pp. 653–661, 2000.

[20] Boehm, Barry W., Software Engineering Economics, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 2006.

Biography

MR.S.PASUPATHY received the B. E degree in Computer Science and Engineering from Government College Of Technology,Coimbatore in April -1990. He received the M.E degree in Computer Science and Engineering from Kongu Engineering College in the year Dec-2000. He worked at Kongu Engineering College,Perundhurai , Erode from 1990 to 2001. He has been with Annamalai University, since 2001. He is pursuing his Ph.D in Computer Science and Engineering at Annamalai University. He published 4 papers in international conferences and journals. His research interest includes Object Oriented Analysis and Design and Software Engineering.