Journal of Theoretical and Applied Information Technology
20th March 2014. Vol. 61 No.2
© 2005 - 2014 JATIT & LLS. All rights reserved·
ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

ANALYZING THE EFFICIENCY OF PROGRAM THROUGH VARIOUS OOAD METRICS

MR. S. PASUPATHY [1] AND DR. R. BHAVANI [2]
[1] Associate Professor, Dept. of CSE, FEAT, Annamalai University, Tamil Nadu, India.
[2] Professor, Dept. of CSE, FEAT, Annamalai University, Tamil Nadu, India.
e-mail: [1]pasuannamalai@gmail.com , [2]shahana_1992@yahoo.co.in

ABSTRACT

Software plays an important role in this today's fast moving world. As everything being computerized, each and every activity must be programmed to lead a successful life. In business environment, to maintain the successful path, many new techniques and technologies have been implemented. Of those, programming for business needs, many new kinds of technologies have been emerged. One of the most useful and easiest technologies to implement is OOAD (Object-Oriented Analysis and Design).
OOAD is most powerful and easy, since the program developed in this object-oriented environment seems to be simple and performs better compared to any other programming language. Object-Oriented technology is becoming increasingly popular in industrial software development environments. Object-Oriented Metrics are the measurement tools adapted to the Object Oriented Paradigm to help manage and foster quality in software development.
In this research paper, we investigate several object oriented metrics proposed by various researchers. Based on the investigation, we propose a new methodology to determine the program efficiency and the quality. Our proposed methodology will produce the result based on the measurement carried out in the programming section.

Keywords: *Business Environment, Object-Oriented environment, Object-Oriented Metrics, OOAD, Software.*

1. INTRODUCTION

The design and development of software using object oriented paradigm is gaining popularity day by day. Object Oriented Analysis and Design of software provide many benefits to both the program designer and the user. Object Orientation contributes to the solution of many problems associated with the development and quality of software product. This technology promises greater programmer productivity, better quality of software and lesser maintenance cost.

Object-Oriented technology is becoming one of the new emerging technologies in this today's computerized world. This technology helps in the development of software product of higher quality and lower maintenance costs. Since the traditional software metrics aims at the procedure-oriented software development so it cannot fulfill the requirement of the object-oriented software, as a result a set of new object oriented software metrics came into existence. Object Oriented metrics are the measurement tools adapted to the Object Oriented paradigm to help manage and foster quality in software development.

Object Oriented Software development requires a different approach from more traditional functional decomposition and data flow development methods. While the functional and data flow approaches commence by considering the systems behavior and/or data separately, object oriented analysis approaches the problem by looking for system entities that combine them. Object oriented analysis and design focuses on objects as the primary agents involved in a computation; each class of data and related operations are collected into a single system entity. There are several object-oriented programming languages that supports object oriented paradigm. Most commonly used are Java, C++, C Sharp, Vb.net. Each programming languages consists of a unique compiler and interpreter to check the program for errors. When the result shows null, then the program is said to be of good quality. Otherwise, it is not so.

The most important skill in Object-Oriented Analysis and Design is assigning responsibilities to objects. That determines how objects interact and what classes should perform what operations. Certain tried-and-true solutions to design problems have been expressed as principles of best practice, often in the form of Design Patterns. A Design Pattern is a named problem solution formula that applies excellent design principles. All software Analysis and Design is preceded by the analysis of requirements. One of the basic principles of good design is to defer decisions as long as possible. The more you know before you make a design decision, the more likely it will be that the decision is a good one.

Analysis:

- Analysis is a broad term. In Software development, we are primarily concerned with two forms of analysis.
- Requirements Analysis is discovering the requirements that a system must meet in order to be successful.
- Object Analysis is investigating the object in a domain to discover information important to meet the requirements

Design:

- Design emphasizes a conceptual solution that fulfills the requirements. A design is not an implementation, although a good design can be implemented when it is complete.
- There are subsets of design, including architectural design, object design, and database design.

Object-Oriented Analysis:

- The emphasis is on finding and describing the objects (or concepts) in the problem domain.
- In a Library Information System, some of the concepts include *Book*, *Library*, and *Patron*.

Object-Oriented Design:

- The emphasis is defining software objects and how they collaborate to fulfill the requirements.
- In a Library Information System, a *Book* software object may have a *title* attribute and a *getChapter* method.

In this research paper, we have to propose a methodology with OOAD Metrics to analysis the program efficiency and to determine the quality of the program. The OOAD Metrics defined in this paper, will evaluate the lines of code in the program to examine the quality of the program.

2. RELATED WORK

In paper [1], 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 [2]), to provide easily measurable, universal source code attributes.

In paper [3], 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 [4], 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 [5], 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 [6], 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.

In paper [7], Chen et al tested the hypothesis that generic recovery techniques, such as process pairs, can survive most application faults without using application-specific information. They examined in detail the faults that occur in three, large, open-source applications: the Apache web server, the GNOME desktop environment, and the My SQL database.

In paper [8], Nair et al. described a case study of combinatorial testing for a small subsystem of a screen-based administrative database. The system was designed to present users with input screens, accept data, then process it and store it in a database. The study was extremely limited in that only one screen of a subsystem with two known faults was involved, but pair wise testing was sufficient to detect both faults. In paper [9], Wallace and Kuhn reviewed 15 years of medical device recall data gathered by the US Food and Drug Administration (FDA) to characterize the types of faults that occur in the application domain. These applications include any devices under FDA authority, but are primarily small to medium sized embedded systems, and would range from roughly 104 to 105 lines of code.

In paper[10], Kuhn and Reilly analyzed reports in bug tracking databases for open source browser and server software, the Mozilla web browser and Apache server. Both were early releases that were undergoing incremental development. In paper [11], Richard stated that Exhaustive testing of computer software was intractable, but empirical studies of software failures suggested that testing can in some cases be effectively exhaustive.

In paper[12], Edgar Gabriel et al pointed that a large number of MPI (Multiple Programming Interface- like Multitasking) implementations are currently available, each of which emphasize different aspects of high-performance computing or are intended to solve a specific research problem. It also presented a high-level overview the goals, design, and implementation of Open MPI.

In this paper, the proposed method has to be developed by consolidating the related papers and also improve with more special features.

3. METHODOLOGY

3.1 Proposed Work

The aim of the research paper is to propose a methodology to detect the program efficiency. The efficiency of the program can be determined by considering the Lines of Code (LOC) in the program and the error rate occurred in the program during the compilation process.

The summary of the research work is described below: In our previous research works, we described the methodology to divide the program into number of sub-programs to identify the number of classes, methods, identifiers and so on. Then the program has to be compiled to detect for errors. From the errors detected and the identified number of modules, the error rate has been calculated. From the calculated error rate, the quality of the program has to be determined.

This research work is the extension of the previous works. In this paper, the efficiency of the program can be determined through successive compilation. From the compilation result, the program quality and the error rate has to be calculated. Based on the calculated result, the efficiency of the program can be determined by considering the Lines of Code (LOC) in the program.

The compilation result has to be stored with various version names depending upon the number of times the process can be carried out. From the result, the comparison can be made out as follows:

If the number of classes increases and the Lines of code increases, the error rate increases, then the program quality is not bad, since the ratio of increase in error rate is directly proportional to the ratio of increase in the program code.

Code α Error_Rate

If the number of classes and the Lines of code remains constant, but the error rate gets increase, then the program is in bad quality, since the ratio of increase in error rate is inversely proportional to the ratio of increase in the program code.

Code α 1/ Error_Rate

Thus based on the comparison ratio, the program efficiency has to be determined and from the determination result, the program is said to be used by the user or to drop the program.

This proposed methodology also consists of an algorithm to determine the efiiciency of the program. The algorithm is given below with proper explanation.

3.2 Algorithm

```
Begin
Create the configuration file
Get the program to analysis as input
Split the program into number of classes and functions using the configuration
file
Initialize the count as zero
Repeat
Compile the program using appropriate compiler
Store the compilation result as 'program_name.version_number'
Increment the count
Until the program becomes error free or user gives END instruction
For i in 1 to count
Read the result from the stored file 'program_name.version_number(i)'
Read the result from the stored file 'program_name.version_number(i+1)'
Read the program named 'program_name(i)'
Read the program named 'program_name(i+1)'
Loc1 = Count the Lines of Code (LOC) in program(i)
Loc2 = Count the Lines of Code (LOC) in program(i+1)
Err1 = Get the error rate from compilation result of program.version(i)
Err2 = Get the error rate from compilation result of program.version(i+1)
If Loc1<Loc2 then
  If Err1<Err2 then
    Program efficiency is good
  End if
Else
  If Loc1=Loc2 then
   If Err1<Err2 then
     Program efficiency is bad
   End if
  End if
End if
Next
End
```

3.3 Algorithm Explanation

The algorithm described above works as follows: The initial step is to develop a configuration file to divide the program into number of classes and functions. Then the program is taken as input and using the configuration file developed, the program is splitted. Then the program is compiled successively until the program becomes error-free or the user gives the END instruction. The compilation result is stored successively with appropriate version. Then the program efficiency is determined by stored result. With the help of the result, the program is read from first compilation result and the error rate is noted. Then the program is read from next compilation result and the error rate is noted. From the two programs, the Lines of Code can be identified and compare it. If the LOC increases with the error rate, then the program efficiency is good. Otherwise, the efficiency is bad. Thus the program efficiency is determined.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

4. EXPERIMENTAL SETUP

The proposed methodology is experimentally verified by implementing the algorithm in a business environment. For the developed program in business, the proposed algorithm is implemented and the efficiency of the program can be determined. First, the program is compiled for its result. From the result, the efficiency is determined through the implemented algorithm. From the experimental result, it is proven that the proposed method performs well.

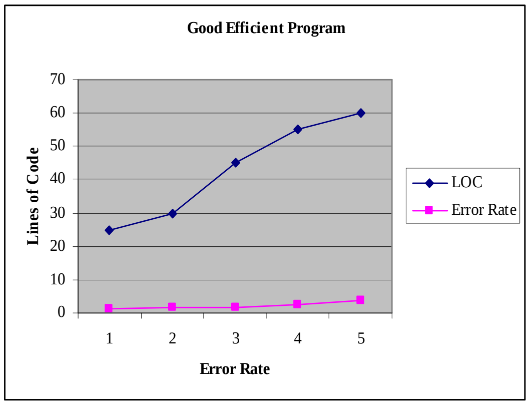

Fig-1: Good Efficiency Program

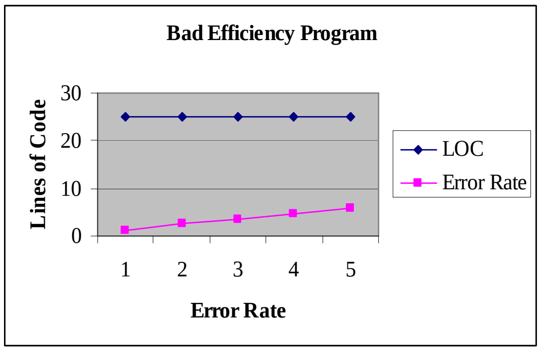

Fig-2: Bad Efficiency Program

5. CONCLUSION

Object-Oriented programming is a modern and powerful technology which will lead a successful path in this today's programming world. This research paper presented some object oriented metrics that will be used to measure the program efficiency. The results of the object oriented metrics is implemented on various experiments and the results shows comparatively good.

However, the metrics presented in this research paper are by no means a complete set of object oriented metrics. But this analysis can be used as a reference by software developers and managers for building a fault free, reliable and easy to maintain software product based on object-oriented technology. So future work will be to refine the current metrics and define additional metric

REFERENCES

[1] M. Halstead, *Elements of Software Science*. New York: Elsevier, 1977.

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

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

[4] 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.

[5] 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.

[6] A. J. Albrecht, "Measuring application development productivity," in *IBM Application Development Symposium*, 1979, pp. 83–92.

[7] Subhachandra Chandra and Peter, M. Chen, Computer Science and Engineering Division, Department of Electrical Engineering and Computer Science, University of Michigan, "Whither Generic Recovery from Application Faults? A Fault Study using Open-Source Software", pp. 97-106, June 2000.

[8] V.N. Nair, D.A. James, W.K. Erlich, and J. Zevallos, "A Statistical Assessment of Some Software Testing Strategies and Application of Experimental Design Techniques," Statistica Sinica, vol. 8, no. 1, pp. 165- 184, 1998.

[9] D.R. Wallace and D.R. Kuhn, "Failure Modes in Medical Device Software: An Analysis of 15 Years of Recall Data," Int'l J. Reliability, Quality and Safety Eng., vol. 8, no. 4,August 2001.

[10] D.R. Kuhn and M.J. Reilly, "An Investigation of the Applicability of Design of Experiments to Software Testing," Proc. 27th NASA/IEEE Software Eng. Workshop, Dec. 2002.

[11] D. Richard Kuhn, Senior Member, IEEE, Dolores R. Wallace, Member, IEEE Computer Society, and Albert M. Gallo Jr., "Software Fault Interactions and Implications for Software Testing" June 2004.

[12] Edgar Gabriel, Graham E. Fagg, George Bosilca, Thara Angskun, Jack J. Dongarra, Jeffrey M. Squyres, Vishal Sahay, Prabhanjan Kambadur, Brian Barrett, Andrew Lumsdaine, Ralph H. Castain, David J. Daniel, Richard L. Graham, Timothy S. Woodall Innovative Computing Laboratory, University of Tennessee, Open System Laboratory, Indiana University, "Open MPI: Goals, Concept, and Design of a Next Generation MPI Implementation". Sep 2004