An Empirical Validation of Code and Design Metrics for Object-Oriented Software at Run-time based on Execution Trace Events

A.Kavitha[1] A. Shanmugam[2]

ABSTRACT

Object-oriented design and development has become popular in today's software development environment. The benefits of object-oriented software development are now widely recognized. Object-oriented development requires not only different approaches to design and implementation; it also requires different approaches to software metrics. Many metrics have been proposed related to various constructs like class, coupling, cohesion, inheritance, information hiding and polymorphism. The metrics for object-oriented systems are different due to the different approach in program paradigm and in object-oriented language itself. Metric data provides quick feedback for software designers and managers. Analyzing and collecting the data can predict design quality. The current research on modeling and measuring the software components through currently available metrics at compile time is insufficient. Traditional coupling measures take into account only "static" measures. They do not account for "dynamic" measures due to polymorphism and may significantly underestimate the complexity of software and misjudge the need for code inspection, testing and debugging. The studies show that dynamic measures also plays vital role in quality measurement. And also, there are quite a few sets of proposed metrics of object-oriented metrics for object-oriented software in the literature and research papers at compile time (static measurement). For dynamic measurement such metrics are not available. This paper addresses the validation of the available metrics which can be used to measure the quality of the software based on the information collected from the execution trace events. The definition of such five different metrics is presented in this paper. The presented metrics are also validated by couple of real projects that use object-oriented language in their projects.

Keywords: OO software metrics, Software quality, Dynamic Metrics, Object, Polymorphism, Coupling, Information-Hiding, Quality measurements

1. INTRODUCTION

Object-oriented design and development has become popular in today's software development environment. The benefits of object-oriented software development are now widely recognized [1]. Object-oriented development requires not only different approaches to design and implementation; it also requires different approaches to software metrics. Metrics for object-oriented system are still a relatively new field of study. The traditional metrics such as lines of code and Cyclomatic complexity have become standard for traditional procedural programs [8,1].

The metrics for object-oriented systems are different due to the different approach in program paradigm and in object-oriented language itself. An object-oriented program paradigm uses localization, encapsulation, information hiding, inheritance, object abstraction and

[1]Lecturer(SS) in Computer Science, Kongunadu Arts & Science College, Coimbatore – 641 029. e-mail : kavithaathi@yahoo.co.in

[2]Principal, Bannari Amman Institute of Technology, Sathyamangalam – 638 401.

polymorphism, and has different program structure than in procedural languages. [8]

Software metrics are often categorized into *product metrics* and *design metrics* [9]. Project metrics are used to predict project needs, such as staffing levels and total effort. They measure the dynamic changes that have taken place in the state of the project, such as how much has been done and how much is left to do. Project metrics are more global and less specific than the design metrics. Unlike the design metrics, project metrics do not measure the quality of the software being developed.

Design metrics are measurements of the static state of the project design at a particular point in time. These metrics are more localized and prescriptive in nature. They look at the quality of the way the system is being built. [9].

Design metrics can be divided into *static metrics* and *dynamic metrics* [11]. Dynamic metrics have a time dimension and the values tend to change over time. Thus dynamic metrics can only be calculated on the software as it is executing. Static metrics remain invariant and are usually calculated from the source code, design, or specification.

There are quite a few sets of proposed metrics of object-oriented metrics for object-oriented software in the literature and research papers. Only few of them can be presented in this document. The presented metric suite in this document is selected from 'A Metrics Suite for Object Oriented Design' article [6] because it describes a basic suite of object-oriented metrics [1] and its metrics has also tested in practice [6,3]. The basic set of metrics contains total of six different metrics that are presented in the Chapter 3. More metrics has been presented for example in the Lorenz's and Kidd's book 'Object-Oriented Software metrics' [9].

In this paper the focus is on *static product metrics* that are described in Chidamber's and Kemerer's article 'A Metrics Suite for Object Oriented Design' [6]. The relationship between different object-oriented metric values is out-of scope in this paper as well as automated tools for collecting object-oriented metric data. This paper doesn't cover measuring design pattern metrics.

2. Rationale For Measurement

The intent of the metrics proposed is to provide help for object-oriented developers and managers to foster better designs, more reusable code, and better estimates. The metrics should be used to identify anomalies as well as to measure progress. The numbers are not meant to drive the design of the project's classes or methods, but rather to help us focus our efforts on potential areas of improvement. The metrics can help each of us improve the way we develop the software. The metrics, as supported by tools, makes us *think* about how we subclass, write methods, use collaboration, and so on. [9]. They help the engineer to recognize parts of the software that might need modifications and re-implementation. The decision of changes to be made should not rely only on the metric values [11].

The metrics are guidelines and not rules and they should be used to support the desired motivations. The intent is to encourage more reuse through better use of abstractions and division of responsibilities, better designs through detection and correction of anomalies. Positive incentives, improvement training and mentoring, and effective design reviews support probability of achieving better results of using object-oriented metrics. [9]

If we are going to improve the object-oriented software we develop, we must measure our designs by well-defined standards. Thresholds are affected by many factors, including the state of the software (prototype, first release, third reuse and so on) and your local project experiences. The language used and different coding styles affect some of the metrics. This is primarily handled with different threshold values for the metrics, which indicate heuristic ranges of better and worse values. For example, C++ tends to have larger method sizes than Smalltalk. Thresholds are not absolute laws of nature. They are heuristics and should be treated as such. Possible problems in our system designs can be detected during the development process. [9]

Software should be designed for maintenance [1]. The design evaluation step is an integral part of achieving a high quality design. The metrics should help in improving the total quality of the end product, which means that quality problems could be resolved as early as possible in the development process. It is a well-known fact that the earlier the problems can be resolved the less it costs to the project in terms of time-to-market, quality and maintenance.

3. Code And Design Metrics Suite

Metric 1: Weighted Methods per Class (WMC)

WMC is a sum of complexities of methods of a class. Consider a Class C_1 with Methods $M_1 \ldots M_n$ that is defined in the class. Let $c_{1 \ldots} c_n$ be the complexity of the methods [6]. Then:

$$\mathrm{WMC} = \sum_{i=1}^{n} c_i$$

WMC measures size as well as the logical structure of the software. The number of methods and the complexity of the involved methods are predictors of how much time and effort is required to develop and maintain the class [11, 6]. The larger the number of methods in a class, the greater the potential impact on inheriting classes. Consequently, more effort and time are needed for maintenance and testing. Furthermore, classes with large number of complex methods are likely to be more application specific, limiting the possibility of reuse. Thus WMC can also be used to estimate the usability and reusability of the class [SyY99]. If all method complexities are considered to be unity, then WMC equals to *Number of Methods* (NMC) metric.

Metric 2: Depth of Inheritance Tree (DIT)

The depth of a class within the inheritance hierarchy is the maximum length from the class node to the root of the tree, measured by the number of ancestor classes. The deeper a class is in the hierarchy, the greater the number of methods it is likely to inherit, making it more complex to predict its behavior. Deeper trees constitute greater design complexity, since more methods and classes are involved. The deeper a particular class is in the hierarchy, the greater potential reuse of inherited methods. [6].

Metric 3: Number of Children (NOC)

Number of children metric equals to number of immediate subclasses subordinated to a class in the class hierarchy. Greater the number of children, greater the reuse, since inheritance is a form of reuse. Greater the number of children, the greater the likelihood of improper abstraction of the parent class. If a class has a large number of children, it may be a case of misuse of sub classing. The number of children gives an idea of the potential influence a class has on the design. If a class has a large number of children, it may require more testing of the methods in that class. [6]. In addition, a class with a large number of children must be flexible in order to provide services in a large number of contexts.

Metric 4: Coupling Between Object Classes (CBO)

CBO for a class is a count of the number of other classes to which is coupled. CBO relates to the notion that an object is coupled to another object if one of them acts on the other, i.e., methods of one uses methods or instance variables of another. Excessive coupling between object classes is detrimental to modular design and prevents reuse. The more independent a class is, the easier it is to reuse it in another application. In order to improve modularity and promote encapsulation, inter-object class couples should be kept to a minimum [6]. Direct access to foreign instance variable has generally been identified as the worst type of coupling [11].

The larger the number of couples, the higher the sensitivity to changes in other parts of the design, and therefore maintenance is more difficult. A measure of coupling is useful to determine how complex the testing of various parts of a design is likely to be. The higher the inter-object class coupling, the more rigorous the testing needs to be. [6,1].

Metric 5: Response For a Class (RFC)

The response set of a class is a set of methods that can potentially be executed in response to a message received by and object of that class[2]. RFC measures both external and internal communication, but specifically it includes methods called from outside the class, so it is also a measure of the potential communication between the class and other classes. [6,1] RFC is more sensitive measure of coupling than CBO since it considers methods instead of classes.

If a large number of methods can be invoked in response to a message, the testing and debugging of the class becomes more complicated since it requires a greater level of understanding required on the part of the tester. The larger the number of methods that can be invoked from a class, the greater the complexity of the class.

4. Evaluation Of Metrics

The discussion in this section is mainly based on Chidamber's and Kemerer's document 'A Metrics Suite for Object Oriented Design' [6]. This section presents only part of the testing details and results that is described in the original document. This section describes also briefly results of the suite of object-oriented design metrics introduced in Chidamber's and Kemerer's document 'A Metrics Suite for Object Oriented Design' [6] which can be used to measure the quality based on the information collected from the execution trace events at run time.

Chidamber and Kemerer who introduced the basic suite for collecting object-oriented code and design metrics tested the metrics suite with two projects. The metrics proposed in their paper were collected using automated tools developed for this research at two different organizations which will be referred to here as Site A and Site B [6].

Site A is a software vendor that uses object-oriented design in their development work and has a collection of different C++ class libraries. Metrics data from 634 classes from two C++ class libraries that are used in the design of graphical user interfaces (GUI) were collected. Both these libraries were used in different product applications for rapid prototyping and development of windows, icons and mouse based interfaces. Reuse across different applications was one of the primary design objectives of these libraries..

Site B is a semiconductor manufacturer and uses the Smalltalk programming language for developing flexible machine control and manufacturing systems. Metrics were collected on the class libraries used in the implementation of a computer aided manufacturing system for the production of VLSI (Very Large Scale Integration) circuits. Over 30 engineers worked on this application, after extensive training and experience with object orientation and the Smalltalk environment. Metrics data from 1459 classes from Site B were collected.

The data from two different commercial projects and subsequent discussions with the designers at those sites lead to several interesting observations that may be useful for managers of object-oriented projects. Designers may tend to keep the inheritance hierarchies shallow, forsaking reusability through inheritance for simplicity of understanding. This potentially reduces the extent of method reuse within an application. However, even in shallow class hierarchies it is possible to extract reuse benefits, as evidenced by the class with 87 methods at Site A that had a total of 43 descendants. This suggests that managers need to proactively manage reuse opportunities and that this metrics suite can aid this process.

Another demonstrable use of these metrics is in uncovering possible design flaws or violations of design philosophy. As the example of the command class with 42 children at Site A demonstrates, the metrics help to point out instances where sub classing has been misused. This is borne out by the experience of the designers interviewed at one of the data sites where excessive declaration of sub classes was common among engineers new to the object-oriented paradigm. These metrics can be used to allocate testing resources. As the example of the interface classes at Site B (with high CBO and RFC values) demonstrates, concentrating test efforts on these classes may have been a more efficient utilization of resources.

Another application of these metrics is in studying differences between different object-oriented languages and environments. As the RFC and DIT data suggest, there are differences across the two sites that may be due to the features of the two target languages. However, despite the large number of classes examined (634 at Site A and 1459 at Site B), only two sites were used in this study, and therefore no claims are offered as to any systematic differences between C++ and Smalltalk environments [6].

Chidamber's and Kemerer's object-oriented metrics showed to be better predictors than the best set of 'traditional' code metrics, which can only be collected during later phases of the software processes. [3].

This empirical validation provides the practitioner with some empirical evidence demonstrating that most of Chidamber's and Kemerer's object-oriented metrics can be useful quality indicators for measuring quality of a software based on run time execution trace events also. Furthermore, most of these metrics appear to be complementary indicators, which are relatively independent from each other.

5. Conclusions

This paper introduces the basic metric suite for object-oriented design. The need for such metrics is particularly acute when an organization is adopting a new technology for which established practices have yet to be developed. The metric suite is not adoptable as such and according to some other researches it is still premature to begin applying such metrics while there remains uncertainty about the precise definitions of many of the quantities to be observed and their impact upon subsequent indirect metrics. For example the usefulness of the proposed metrics, and others, would be greatly enhanced if clearer guidance concerning their application to specific languages were to be provided. [ChS95]

Metric data provides quick feedback for software designers and managers. Analyzing and collecting the data can predict design quality. If appropriately used, it can lead to a significant reduction in costs of the overall implementation and improvements in quality of the final product. The improved quality, in turn reduces future maintenance efforts. According to my opinion it's motivating for the developer to get early, continuous(dynamic) feedback about the quality in design and implementation of the product they develop and thus get a possibility to improve the quality of the product as early as possible. It could be a pleasant challenge to improve own design practices based on measurable data.

It should be also kept in mind that metrics are only guidelines and not rules. They are guidelines that give an indication of the progress that a project has made and the quality of design .

References

[1] Alkadi Ghassan, Carver Doris .L, *"Application of Metrics to Object-Oriented Designs"*, Proceedings of IEEE Aerospace Conference, Vol. 4, PP. 159 -163, March 1998.

[2] Archer Clark, Stinson Michael, *"Object-Oriented Software Measures"*, TechnicalReport CMU/SEI-95-TR-002, ESC-TR-95-002, Software Engineering Institute, Carnegie Mellon University, April 1995.

[3] Victor R Basili, Fellow, IEEE, Lionel C Briand, Walcélio L, Melo, Member IEEE Computer Society, *"A Validation of Object-Oriented Design Metrics as Quality Indicators"*, IEEE Transactions on Software Engineering, Vol. 22, No. 10, PP. 751 - 761, October 1996.

[4] Beck Kent, *"Extreme Programming Explained: Embrace Change,"* Addison-Wesley, PP. 190, 2001.

[5] Lionel C. Briand, Sandro Morasca, Member, IEEE Computer Society, Victor R. Basili, Fellow, IEEE, *"Defining and Validating Measures for Object-Based High-Level Design"*, Vol. 25, No. 5, PP. 722 - 743, September/October 1999.

[6] Chidamber Shyam R, Kemerer Chris F, *"A Metrics Suite for Object Oriented Design"*, IEEE Transactions on Software Engineering, Vol. 20, No. 6, PP. 476 - 493, June 1994.

[7] Gamma Erich, Helm Richard, Johnson Ralph, Vlissides John' *"Design Patterns CD, Elements of Reusable Object-Oriented Software"*, Addison Wesley Longman, Inc.1998.

[8] Shuqin Li-Kokko, *"Code and Design Metrics for Object-Oriented Systems"*, Helsinki University of Technology, PP. 9, 2000.

[9] Lorenz Mark, Kidd Jeff, *"Object-Oriented Software Metrics: A Practical Guide"*, P T R Prentice Hall, Prentice-Hall, Inc. A Pearson Education Company, PP. 146, 1994.

[10] Metrics for Analysis and Improvement of Software Architectures (MAISA), Research and Development Project at the University of Helsinki in Department of Computer Science, Paakki Jukka, Verkamo Inkeri, Gustafsson Juha, Nenonen Lilli, 1999 - 2001.

[11] Tarja Systä, Ping Yu, *"Using OO Metrics and Rigi to Evaluate Java software"*, University of Tampere, Department of Computer Science, Series of Publications A A-1999-9, PP. 24, July 1999.

Author's Biography

Dr. A. Shanmugam, received the Bachelor of Engineering Degree in Electronics and Communication Engineering from PSG College of Technology, Coimbatore, Madras University, India in the year 1972 and Master of Engineering Degree in Applied Electronics from College of Engineering, Guindy, Chennai, Madras University, India in the year 1978. He was awarded Doctor of Philosophy (Ph.D.) in Computer Networks from PSG College of Technology, Coimbatore, Bharathiyar University, India in the year 1994. Currently he working as Principal, Bannari Amman Institute of Technology (BIT), Sathyamangalam, Erode District, Tamilnadu, in May 2004 and ever since a lot of changes have been introduced in which made BIT one among the top ranking institutions of Anna University. He awarded **"Bharatiya Vidya Bhavan National Award for Best Engineering College Principal"** for the year 2005 by the Indian Society for Technical Education, New Delhi. His specialization is computer networks. He has guided Eight Ph.D.,Scholars. Now also, he is guiding several Ph.D., Scholars. He has presented and published papers in various national / international conferences and journals.

Mrs. A. Kavitha, obtained her B.Sc., (Maths) degree in the year 1993 and M.C.A. degree in the year 1996 from Vellalar College for Women (Autonomous), Erode District, Tamilnadu. She awarded Master of Philosophy (M.Phil.,) in Computer Science from Erode Arts College (Autonomous), Erode in the year 2004. She is working as a Lecturer in Computer Science Kongunadu Arts and Science College (Autonomous), Coimbatore - 29. At presently she is doing her Ph.D., research work in Vinayaka Missions University, Salem, India under the Guidance of Dr. A. Shanmugam. She has presented various papers in National and International Conferences. One of her research article is published in IEEE explore in January 2008 issue. Another paper is also accepted for International Conference, IMECS 2008, held on March 2008 in Hong Kong. Her interested research fields are: Software engineering, Distributed Data Bases.