OOCA: Testing of Object Oriented Developed Software

Prof. Mostafa Sami Mahmoud
Computer Science Professor
Dean Faculty of Information Technology , Misr University for Science & Technology
mostafa_sami2002@yahoo.com

Dr.Aliaa A.A. Youssif
Faculty of Computers and Information, Helwan University, Cairo, Egypt
aliaay@yahoo.com

Ahmed M.Abd El- Zaher
Faculty of Computer Sciences, Misr International University, Cairo, Egypt
aazaher@email.com

Abstract

This paper addresses the development and implementation of a new automated data collection and measuring tool to test thoroughly any visual basic.net object oriented software design. For this purpose, the study introduces a set of six context coverage metrics in conjunction with traditional structural coverage metrics (a total of nine) in an integrated measuring scheme for testing phase over the object oriented software. The developed system ensure that all object oriented software elements are fully tested including cyclomatic complexity, weighted methods per class, response for a class, lack of cohesion methods, coupling between objects, depth of inheritance tree, number of children, lines of code, and comment percentage.

To establish the confidence in the proposed measuring scheme, this study is applied to two different object oriented software environment namely: an HTML editor and a VHDL design visual basic.net software. Test results are given as a full report illustrating qualitatively the nine criteria of the object oriented software design

1. Introduction

The use of structural coverage metrics to measure the thoroughness of a test set is a well-understood technique. However, the application of the technique to object oriented software is still presenting new challenges.
Many publications were reported before relevant to measure the quality of object oriented software [1-5]. The concepts of software metrics are also well established, and many metrics relating to product quality have been developed and used [6-10].
Traditional structural coverage metrics [1-3] such as statement coverage, branch coverage and condition coverage measure how well the bodies of each method have been tested. Unfortunately these traditional metrics do not take into account all features of the object-oriented design [3]. For instance, the use of polymorphism encapsulation of state-dependent behaviour behind well-defined class interfaces is effectively ignored. Since encapsulations of behaviour are major features of any object-oriented design, metrics which ignore them are insufficient for determining whether the software under test has been thoroughly tested. Therefore, traditional structural coverage metrics are inadequate measures of test thoroughness for object-oriented software systems. New object-oriented coverage metrics are required to ensure thorough testing.
In this study, a set of six specific context coverage metrics [2-3] are used, in conjunction with the traditional structural coverage metrics to develop an automated measuring scheme to ensure that all object oriented software elements are fully tested including cyclomatic complexity, weighted methods per class, response for a class, lack of cohesion methods, coupling between objects, depth of inheritance tree, number of children, lines of code, and comment percentage. The proposed testing system is established based on the latest visual basic.net product as an effective tool for object oriented design. It should be noted that object-oriented metrics evaluation criteria must be able to focus on the combination of functions and data as an integrated object in any object oriented software design. In addition, the evaluation of the utility of a metric as a quantitative measure of software quality must relate to the software assurance technology centre (SATC) Software Quality Model [2]. The selected object-oriented metrics criteria in this paper, therefore, are the evaluation of the following areas:

- *Efficiency of the implementation of the design*
- *Understandability/Usability*
- *Testability/Maintenance*
- *Complexity*
- *Reusability/Application specific*

However, whether a metrics is “traditional” or “specific”, it must be effective in measuring and evaluating the above areas.

2. Object Oriented Testing

Unique features of object-oriented programming and design impose added complexities on the measuring process including message passing, inheritance, and polymorphism. These features require a suite of measures designed to handle them.

The classical strategy for testing computer software begins with "testing in the small" and works outward toward "testing in the large." Stated in the jargon of software testing, we begin with unit testing, then progress toward integration testing, and culminate with validation and system testing [5-10]. In conventional applications،

unit testing focuses on the smallest compliable program unit—the subprogram (e.g., module, subroutine, procedure, component). Once each of these units has been tested individually, it is integrated into a program structure while a series of regression tests are run to uncover errors due to interfacing between the modules and side effects caused by the addition of new units. Finally, the system as a whole is tested to ensure that errors in requirements are uncovered [9, 10].

Traditional metrics used before in conventional object oriented programs are well understood and established in many reports before. Therefore, the attention given here will be focused on the six specific metrics suggested to build up an integrated measuring scheme for object oriented system. This study, however, supports also the use of well known three traditional metrics usually adopted to measure the methods implemented in object oriented software design.

3. Object-oriented specific metrics

Many different metrics have been proposed for object-oriented systems. The object-oriented metrics chosen in this study measure main structures that, if improperly designed, affect negatively the design and code quality attributes. The selected object-oriented metrics are primarily applied to the concepts of classes, coupling, and inheritance.

It should be noted that as with traditional metrics, researchers and practitioners have not reached a common definition or counting methodology. In some cases, the counting method for a metric is determined by the software analysis package being used to collect the metrics.

3.1. Metric 1: Weighted Methods per Class (WMC)

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). 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 [2,3]. 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.

3.2. Metric 2: Response for a Class (RFC)

The RFC is the cardinality of the set of all methods that can be invoked in response to a message to an object of the class or by some method in the class. This includes all methods accessible within the class hierarchy. This metric looks at the combination of the complexity of a class through the number of methods and the amount of communication with other classes. The larger the number of methods that can be invoked from a class through messages, the greater the complexity of the class. If a large number of methods can be invoked in response to a message, the testing and debugging of the class becomes complicated since it requires a greater level of understanding on the part of the tester. A worst case value for possible responses will assist in the appropriate allocation of testing time [2 , 3]. This metric evaluations system design as well as the usability and the testability.

3.3. Metric 3: Lack of Cohesion of Methods (LCOM)

LCOM measures the degree of similarity of methods by instance variable or attributes. Any measure of separateness of methods helps identify flaws in the design of classes. There is a standard way to compute the cohesion:

Calculate for each data field in a class what percentage of the methods use that data field. Average the percentages then subtract from 100%. Lower percentages mean greater cohesion of data and methods in the class [2, 3].

Methods are more similar if they operate on the same attributes. Count the number of disjoint sets produced from the intersection of the sets of attributes used by the methods.

High cohesion indicates good class subdivision. Lack of cohesion or low cohesion increases complexity, thereby increasing the likelihood of errors during the development process. This metric evaluates the design implementation as well as reusability.

3.4. Metric 4: Coupling Between Object Classes (CBO)

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 reused 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 [2,3]. This improves modularity and promotes encapsulation. CBO evaluates design implementation and reusability.

3.5. Metric 5: 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 and is measured by the number of ancestor classes. The deeper a class is within the hierarchy, the greater the number 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, but the greater the potential for reuse of inherited methods. A support metric for DIT is the number of methods inherited (NMI). This metric primarily evaluates reuse but also relates to understandability and testability [2, 3].

3.6. Metric 6: Number of Children (NOC)

The number of children is the number of immediate subclasses subordinate to a class in the hierarchy. It is an indicator of the potential influence a class can have on the design and on the system. The greater the number of children, the greater the likelihood of improper abstraction of the parent and may be a case of misuse of sub classing. But the greater the number of children, the greater the reuse since inheritance is a form of reuse. If a class has a large number of children, it may require more testing of the methods of that class, thus increase the testing time. NOC, therefore, primarily evaluates testability and design [2, 3].

4. Development Object Oriented Code Analysis (OOCA)

The above specific six measuring metrics in addition to the traditional metrics are implemented together in automated object oriented code analysis software established in this research. This developed measuring and code analysis program is designed specially to serve the V.Basic.NET developers in general and V. Basic.NET developers in organizations that develop N systems applications in particular.

It should be noted that Visual Studio .NET is a complete set of development tools for building ASP Web applications, XML Web services, desktop applications, and mobile applications. Visual Basic .NET, Visual C++ .NET, and Visual C# .NET all use the same integrated development environment (IDE), which allows them to share tools and facilitates in the creation of mixed-language solutions.

This study, however, selected the Visual Basic.Net as the main tool of building the automated OOCA scheme for measuring the quality of object oriented software design. The reasons are obviously clear as the Visual Basic.Net is the latest version of the visual basic product family and testing the N systems applications of visual basic.net are sensible selection for this research.

Recently, N systems became a new trend in some organizations that develop more than one copy of product with different programming approaches to solve the same problem with the same software tool depending on the foundation of more than one coding team. These N systems used to minimize the risk of failing for any working software.

The OOCA automated measuring scheme suggested in this paper, should serve the V.Basic.NET developers by giving them an over view of all classes, attributes and methods they used in developing any V.B application. The output test results will guide the coder to have the ability to change in the code in a more efficient faster way. From this point of view and the change that the developer will do inside the code, the testing process will become easier for the tester.

In the N systems developing, the proposed OOCA and evaluating scheme will have another benefit as to analyze the Visual Basic source code for the N software copies. Based on the results of this analysis, a comparison can be made between the multiple copies to select the more efficient module code from the multiple software copies to have the more efficient product source finally available to the client.

5. Applications

To establish the reliability of the proposed testing scheme, the OOCA was applied to two different object oriented applications designed based on Visual Basic.Net software. These two applications are: a VHDL design and HTML editor. Test results are given as a full report illustrating qualitatively the five criteria of the object oriented software design.

5.1. Product No. 1: VHDL Application

The very high descriptive language (VHDL) is a software design package for developing flexible control on machines and manufacturing systems. The OOCA was implemented on a VDHL application designed especially for computer aided manufacturing system for the production of printed circuits.

5.2. Product No.2: HTML Editor

HTML Editor 1.0 is a complete application that allows you to create html file. It has many functions, like create new file, open, save, clear, print, undo, redo, cut, copy, paste, delete, select all, insert date, time, image, find, add tables[first Column, New Column, Cells, More Column] , add fonts [Font Name, Size, Color, Styles] , Spell check, and many more. This application is provided with pop up menu with the facilities of previewing the source code of the editor itself.

The OOCA was successfully applied to these applications and the measuring results were generated.

Figures (1-5) shows the OOCA interface and samples of the output test results of the VHDL product, while as Figures.(6-8) show the output results of the HTML editor based on the proposed scheme of this study.

6. Discussion of Results

The following tables and samples summarize the discussion and the emphasized remarks on the results:

6.1. VHDL Application:

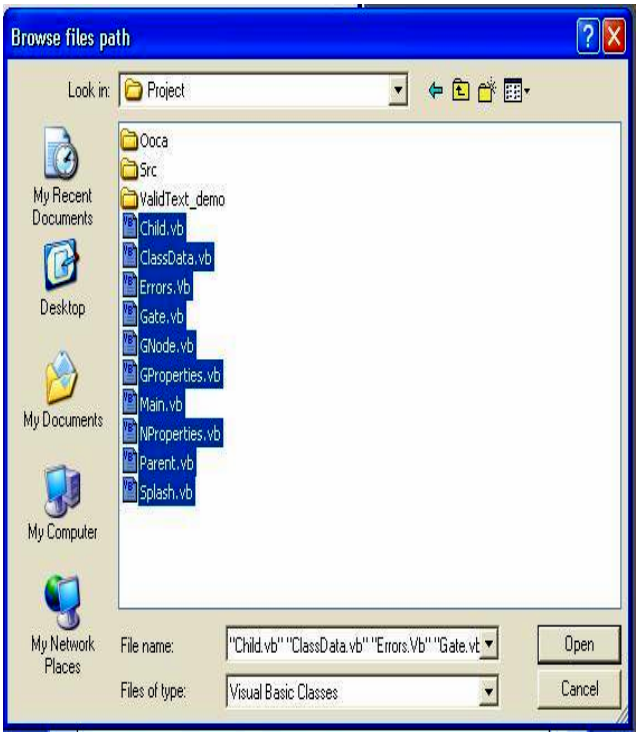

Figure 2: OOCA select files of VHDL Application

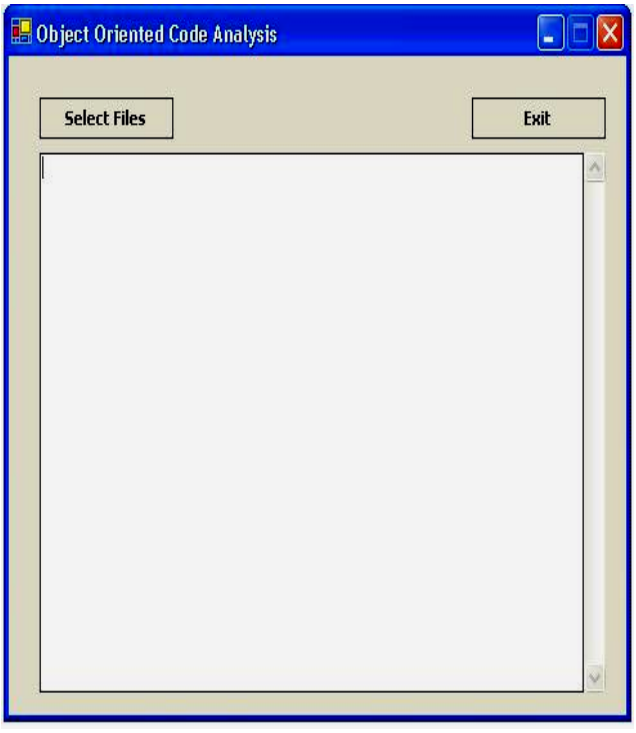

Figure 1: OOCA interface

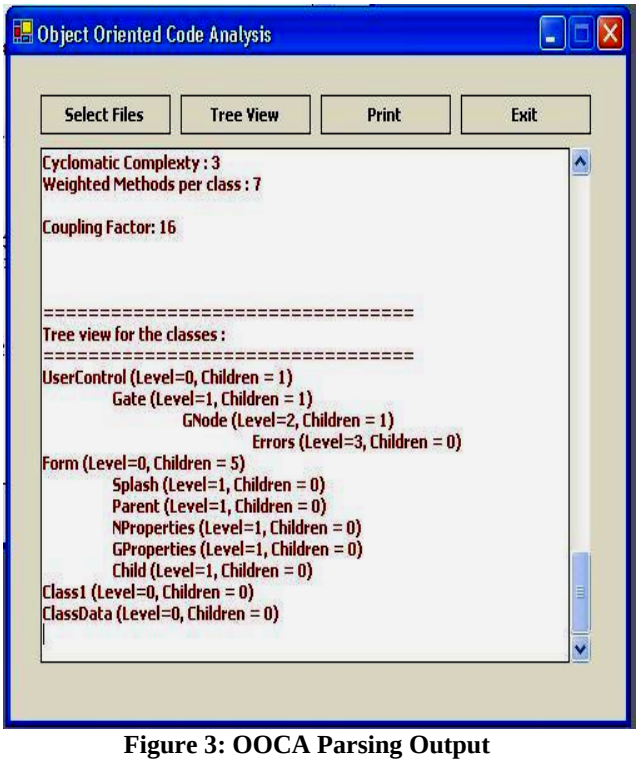

Figure 3: OOCA Parsing Output

Figure 4: OOCA Parsing Tree Output

Figure 5: OOCA Tree View

WMC	There are some classes with suitable number of methods, but there are other classes with more than 45 methods. This can be enhanced by dividing those classes into 3 or 4 subclasses.
DIT	The depth of inheritance values are good in all classes specially the maximum value which is 3. These values are very good compared to the product functionality.
CBO	The results of coupling between objects: The above report indicates no coupling in 3 main classes between any object inside them. On the other hand, in the other 5 classes the coupling factor is so large this means if there is any change in any object we will notice relevant changes in the other objects.
LCOM	The cohesion values are almost perfect because there is almost no relation between methods inside classes.
NOC	Number of children in this product is good because there are two levels in the tree of the product. This tree contains eight classes and therefore the NOC will not be a problem.

6.2. HTML Editor:

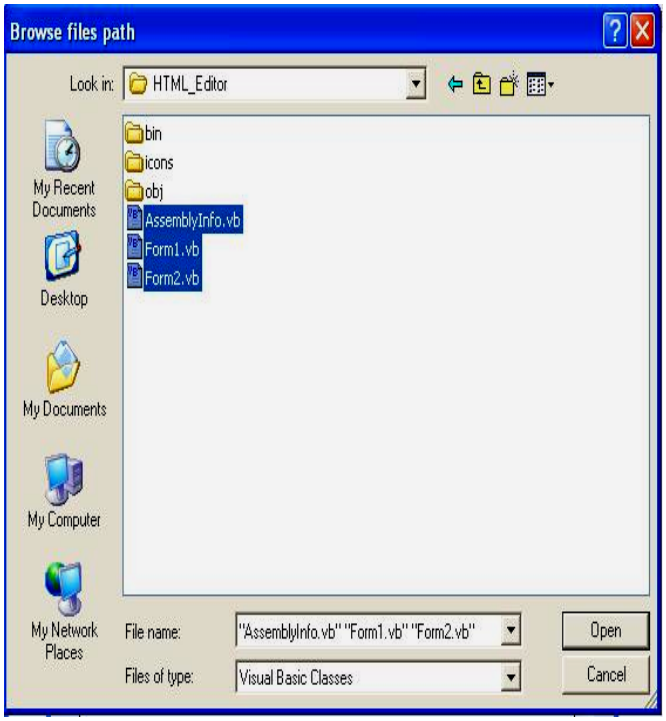

Figure 6: OOCA select files of HTML Editor

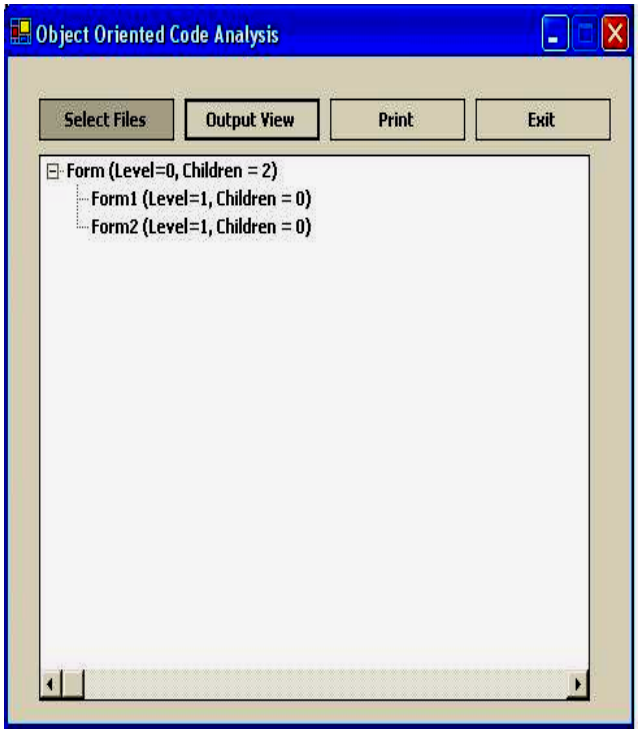

Figure 7: OOCA Output of HTML Editor

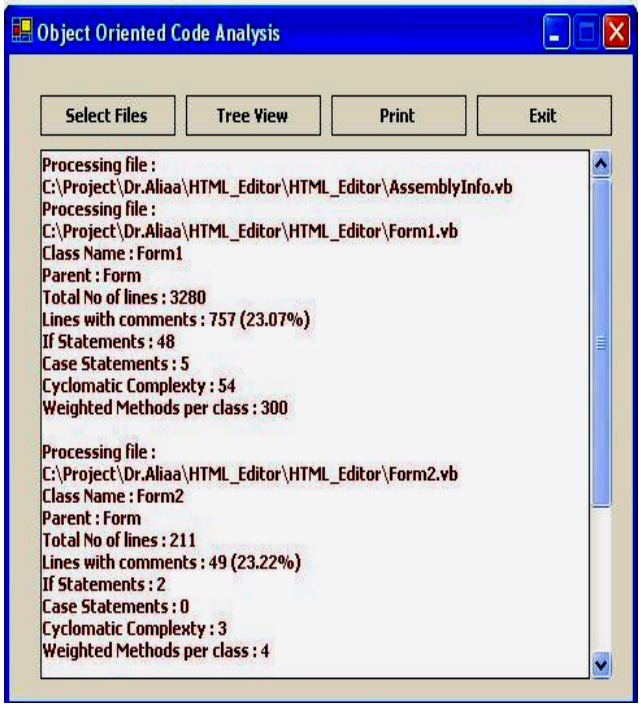

Figure 8: OOCA Tree View of HTML Editor

WMC	There is good methods division between classes and there is no need for enhancing the methods per classes specially that the classes contain 3 or 4 methods at maximum.
DIT	The depth of inheritance values is good in all classes specially that the maximum value is 2 which are very satisfactory compared to the product functionality.
CBO	The analysis report indicates that there is coupling between objects in classes and this is almost the weakness point in this product.
LCOM	There is almost no cohesion between methods in classes because in each class there are 3 methods.
NOC	Number of children in this product is good because there is one level in the tree of the product. So the NOC will not be a problem through the evaluating process of the product.

7. Conclusion

It is confirmed that Object-oriented design and development became more popular in today's object oriented applications and environment. Therefore these developments require a different approach not only to design and implementation, but also to thoroughly measuring tests and evaluations.

Since object oriented technology uses objects and not algorithms as its fundamental building blocks, the approach to software testing metrics for object-oriented programs must be different from the standard testing metrics set.

Some testing metrics, such as lines of code, have become accepted as "standard" for traditional functional/ procedural programs, but for object-oriented systems this is insufficient.

This study established more proper testing scheme for object oriented software design based on nine measuring metrics rather than the three traditional metrics used before.

The applications of the automated OOCA presented in this paper in conjunction with the obtained test results indicate the successful efficiency and reliability of the proposed scheme and this application will be a helper tool for the programmers to be used through the coding phase. As the coders used this application, they will have an over view for the object oriented techniques they used. So they can simplify the usage of these techniques which give the testers the opportunity to make simple test cases.

8. References

[1] Cherniavsky, J.C. and Smith, C.H., "On Weyuker's Axioms for Software Complexity Measures", IEEE Transactions on Software Engineering, vol. 17, pp. 636-638, 1991.

[2] Rosenberg, Linda, “Software Quality Metrics for Object Oriented System Environments”, National Aeronautics and Space Administration, NASA, 1995.

[3] Rosenberg,jhon and Kolling, Micheal,”Testing object orinted programs:Macking it simple”,Proceding of the twenty-eighth SIGCSE technical symposium on computer science education,pp. 34-37 ,1997.

[4] Chen , Yan H. , Tse ,T. H. and Chan , F. T., “In black and white: an integrated approach to class-level testing of object-oriented program.”, ACM Transactions on Software Engineering and Methodology, pages 250–295, 1998.

[5] Bach, James, “ Heuristic Risk-Based Testing”, Software Testing and Quality Engineering Magazine, 1999.

[6] Binder, Robert V., Testing Object Oriented Systems, Modules, patterns and tools, Addison-Wesley Publishing Company, 2000.

[7] Labiche Y., Fosse, Thévenod P. and Waeselynck H., "Testing levels for object-oriented software", Proceedings of the 22nd international conference on Software engineering, 2000.

[8] Somerville, Ian, Software Engineering, Addison-Wesley Publishing Company, 6’Th edition, 2000.

[9] Chen , Yan H. , Tse ,T. H. and Chan , F. T., “ TACCLE: a methodology for object-oriented software testing at the class and cluster levels”, ACM Transactions on Software Engineering and Methodology (TOSEM), Volume 10 Issue 1, 2001.

[10] Martin, Robent C., Agile Software Development, Principles, Patterns and Practices, Prentice Hall Publishing Company, 2003.