International Journal of Computers & Technology
Volume 3 No. 3, Nov-Dec, 2012

Modeling an Object Oriented for Maintenance Purposes

Hamed J. Al-Fawareh
Faculty of Science aand IT
Zarqa University
Zarqa, Jordan

ABSTRACT: *Software maintenance is the last phase of the software life cycle. The aim of the software maintenance is to maintain the software system in accordance with advancement in software and hardware technology. In this paper, we discuss a maintenance system for object-oriented techniques. The paper therefore discusses about a problems in object oriented techniques under the maintenance environment. These problems include understanding object oriented system, complex dependencies in object-oriented system, inheritance, polymorphism and dynamic binding problem that maintainers and developers commonly face. Finally, we talk about the proposed object-oriented maintenance tool.*

1.0 INTRODUCTION

Starting from the last ten years, the use of object-oriented programming in software development has been increasing. Object oriented software systems have been applied in many difficult, complex applications, and in different environments. Each object oriented software system may contain thousands of objects within thousand of classes, a fact, which makes it difficult for maintainers and developers to understand. Furthermore, the object-oriented technique contains difficult entity relationships that are classes relation, inheritance, polymorphism and dynamic binding.

Nowadays, object-oriented technique receives more attention in order to help maintainers and developers understand the object-oriented software system. In this paper, we will highlight some relevant comments from others maintainers.

To assist the maintainers in the previous techniques, we need information such as data flow, control flow, data dependence and control dependence. [7][9] For program understanding and assistance in locations of source errors, we use data dependence and control dependence information to identify all statements in a program including each error object boundaries. Both control flow and data flow information are used to determine the effects of the call reference parameters and global variables. [7][19][20][6]

During an object-oriented program process, a set of objects represent each entity of problem domain and each object has some attributes and operations that will be performed on it. In many object oriented programming languages, objects are grouped into classes.

The object-oriented programming maintenance faces a lot of problems. These problems include software understanding, complex dependencies in object-oriented system, inheritance, polymorphism and dynamic binding problems. Detailed discussions of these problems will be provided later. In an attempt to solve these problems, we will discuss a proposed Object Oriented Maintenance System (OOMS). In the OOMS, we will discuss the proposed solution for helping maintainers to understand, trace and remove the redundant information in object oriented software system.

In addition, we will discuss the object oriented software maintenance environment. In section two, we will go through the object oriented program, and will explain the entities of the object oriented software, that is class, inherits, message and method. In section three, we will discuss the problems facing maintainers of object-oriented software. In section four we will discuss a proposed object oriented maintenance system. Finally we will discuss a conclusion.

2.0 BASIC CONCEPTS OF OBJECT ORIENTED TECHNIQUE

Software engineering and information system used object oriented in many areas of software engineering. There are programming languages, design methodologies, user interface databases and operating systems that have been described as object oriented. A system based on objects is one whereby a computation is represented by a series of entities, which interact to achieve the desired affect.

- *An object:* Object oriented language uses objects as a key to understanding object-oriented technology. An object is a thing to be implemented in application domain. For example, a postgraduate student can represent an object.
- An object has a set of attributes. These attributes define an object state (everything that the software objects knows) and object behavior (everything that the software objects can do). For example a postgraduate student state is ‘name’, ‘metric No.’, ‘program’, ‘faculty’ and ‘nationality’ and his behavior like ‘field of study’, ‘course No.’, ‘add course’, ‘dorp course’ and so on.
- *Classes:* The objects are includes of classes. The class determines everything about an object. When you create a class, you are already, creating an object of that type and the system allocates memory for the variables, which are declared in the class. The main benefit of using a class and object are reusability and modularity and information

hiding from the objects. Software programmers use the same class, and thus the same code, over and over again to create many objects.

- *Subclasses, Superclasses, and Inheritance:* In object-oriented programming languages, classes can be sub or super classes. The subclass is derived from other class and the superclass is that from which, classes are derived. The subclass inherits from the superclass its state and behavior. The subclass can just use the items inherited from its superclass or the subclass can modify or override it. So, the classes become more and more specialized in the hierarchy classes representation.
- *Messages:* Object oriented software usually contains more than one object. A single object alone is generally not very useful in the large software system or application. Software objects interact and communicate with each other by sending messages to each other.
- *Polymorphism* is the ability to take more than one form. An attribute may have more than one set of values and an operation may be implemented by more than one method. For example, a graduate student may work as research assistant, in this case the student is under two superclasses, that is, the graduate student and employee classes.
- *Dynamic Binding* is a method that implements an operation that is unknown until runtime. It is an effective mechanism to implement polymorphism. In another word, an operation may have more than one implementation, the choice of which implementation to use when an operation is invoked is determined at runtime according to the types, the number of arguments and/or the function pointed to by a function pointer. [5]

3.0 OBJECT ORIENTED SOFTWARE MAINTENANCE ENVIRONMENT

The rapid increase in the use of object-oriented techniques in software product and the power of the new technique feature a new set of object oriented software maintenance tools. [5] Program maintenance is an expensive process whereby an existing program is modified for variety of reasons, including corrective maintenance, adaptive maintenance, enhancement and improvement efficiency. [1][4]

An object-oriented technique is easier to reuse than the other programming technique, because object-oriented programming is represented by a set of objects. An object may derive into sub-objects and then will express relations between them. [2][17][8]

The new technique and the object oriented programming constructions make the software maintenance face a lot of difficulties. Some of these difficulties that maintainers should take into account are the following;

1. Software understanding problems.
2. Complex dependencies in object-oriented system, classes problems.
3. Inheritance, polymorphism and dynamic binding problems.

3.1 SOFTWARE UNDERSTANDING PROBLEM

It is easy to understand the relation between functions in a program, definitions and uses of variables, even finding the definition of a function given its name or a call site. But it becomes more complex with huge and complex systems. By ordering the programming understanding problem, object-oriented techniques may not make programs easier to understand, because of complications from inheritance, dynamic binding and a large number of small methods dispersed in the programs. In order to understand object oriented programs, maintainers have to trace the calling relationships in the programs. Such tracing is time consuming and error prone. [14][10]

Several existing tools can generate the module calling hierarchy or structure chart. Calling hierarchies are useful tools for approaches in which the main packaging unit is the processing module (e.g. function or procedural). [12] In the call graph the nodes represent individual procedures and the edge represents the call sites. Since procedure may call another at many points, a call graph may be a multi graph with more than one edges connecting any two nodes. A program's call graph can be constructed efficiently [17] and used for many different applications. Since a call graph represents the entities of an object-oriented program and illustrates the calling relationships among entities, it is useful for program understanding during maintenance. It is also useful for data flow analysis. A call graph is more complex and very hard to represent in the object oriented program, the reasons being dynamic binding, and object-oriented programming constructions such as a class, methods, message and inherits.

3.2 COMPLEX DEPENDENCIES IN OBJECT-RIENTED SYSTEM PROBLEM

In the structured programming there are a lot of dependencies represented as $X \rightarrow Y$, such that a programmer's modifying X must have possible effect on Y. The main types of dependencies are data items (or variables), processing modules and data types. [8]

Dependencies are classified as follows:

- Data dependencies between two variables.
- Calling dependencies between a module and the variables it computes.

- Definitional dependencies between a variable and its type.

In object oriented languages we need to add the following types of entities:

- Object class.
- Methods(which are specific code segments)
- Messages (which may be thought of as "name" of methods)

There are a lot of relations between these entities, which include relations between two classes, as shown in figure 3. The relations between any of two entities (class, message, method and variable) are illustrated in figure 4.

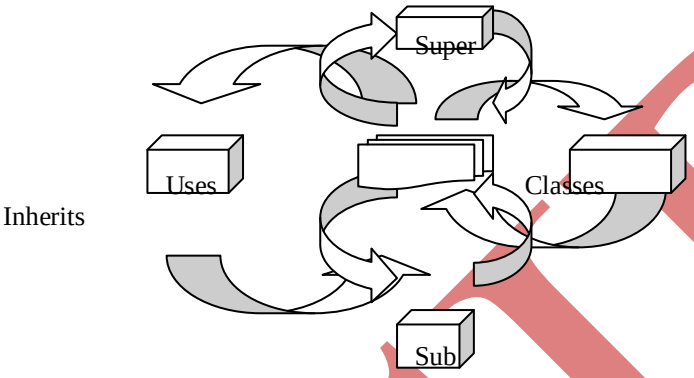

Figure 3: Class Relations

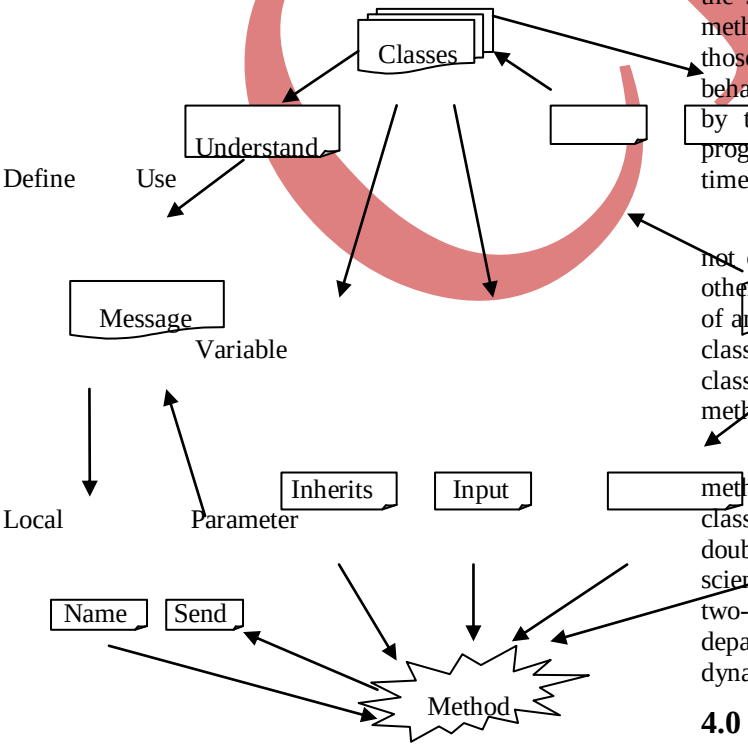

Figure 4: Object Oriented Entities Relation.

The complex relationships that exist in object-oriented system are the main cause of the dependency problem. These relationships as explained above are inheritance, dynamic binding, classes, and nested class. Also, these relationships may include indirect relations, for example when one class imposes indirect dependence on other classes. The traditional maintenance techniques for the structure programming are inadequate for the new object oriented programming, since they do not take into account the complexity of object oriented programming. When we modify one class, we soon found out that we have to understand and trace many other classes in order to make modification, enhancement, or correction.

3.3 INHERITANCE, POLYMORPHISM AND DYNAMIC BINDING PROBLEMS

Object oriented software defines an object as terms of classes. However, object-oriented programming takes this a step further and allows classes to be defined in terms of other classes. Each subclass inherits state from the superclass. Also, each subclass inherits methods from the superclass. However, subclasses are not limited to the state and behaviors provided to them by their superclass. Subclasses can add variables and methods to the ones they inherit from the superclass. Subclasses can also override inherited methods and provide specialized implementations for those methods. Subclasses provide specialized behaviors from the basis of common elements provided by the superclass. Through the use of inheritance, programmers can reuse the code in the superclass many times.

In object oriented language variables are used not only in the class's scope but may also execute in other classes. The variable may refer to another objects of any class using polymorphism. Furthermore, a given class can use a method which is declare in another class, when a given message is sent to execute this method. [22][14]

Object Oriented programming allows different methods for different purposes. Furthermore an object class may have more than one superclass, for example a double major lecture under the department of computer science and mathematics. In this case the lecture has two-superclasses computer science mathematics department. This ability is called polymorphism and/or dynamic binding.

4.0 PROPOSED OBJECT ORIENTED MAINTENANCE SYSTEM (OOMS)

Software systems have been applied in many difficult and complex applications, from different environments. Each software system may contain thousands of source code lines, a fact, which makes it difficult to manually know all the overlapping between the classes, methods, variables and parameters of this software without aid tools. These problems become even more complicated when the maintainer uses a large software system. Maintenance environment helps maintainers and developers by giving them a full information about the software system. This information includes, class relationships, inheritance, polymorphism, call graph, data structures, cross-reference compiler list, and control flow diagram.

The maintainer spends a lot of time and resources in order to understand the object-oriented software system process by reading a class relationships, inherence, polymorphism, tracing and manually comparing thousands of source code lines in the object-oriented software system. OOMS will assist maintainers to understand software systems with different abstraction levels, and maintenance work. OOMS will also assists maintainers and developers to understand the behaviors, external and internal design structures, class relationships, polymorphism, inheritance and the implementations of an object-oriented software system, as well as the process of software system. [6]

The proposed OOMS proceeds by reading a Java source code, extracting the information from the object program and transferring it to the special repository. This information include entities relationships, cross-reference, type abstractions and object oriented dependencies. OOMS uses different repositories for different purposes, also this propose system include lexical analyzer, syntax analyzer for checking the language grammar and removing a syntax errors. Furthermore it contains an interface and user query.

In object oriented program the calling hierarchy would be a hierarchy of methods, and this faces a lot of problems. These problems include the dynamic binding problem, which may make the hierarchy difficult to compute. There may be no real "main" method in the system. This is one of the facts about object oriented design that beginner tends to find disconcerting. Hierarchy of methods loses sight of the grouping of methods in objects, which is presumably the most important aspect of the design. The proposed OOMS represents the object oriented call hierarchy as *Display Object Relationships*. The Display Object Relationships helps maintainers by giving them the relationships between the object oriented technique entities, for example the relations between the classes as a sub, super, and inherits.

This proposed system should help a maintainer in understanding the complex relationships and the complex dependencies among the various object-oriented programming components. The complex dependencies between the classes in an object-oriented programming imply the impact change through the software system. Understanding the complex relationships and dependencies between the object-oriented technique components helps maintainers and developers to understand the object oriented software system.

A new proposed object oriented software maintenance system helps maintainers and developers to remove the redundant information in object oriented software system and save space. Furthermore, software maintenance system gives maintainers and developers tracing facility, detailed information about the overlapping existing between the classes in their program and takes into account the polymorphism, hiding information and dynamic binding problems.

5.0 CONCLUSION

Understanding object-oriented program, complex dependencies, Inheritance, dynamic binding and polymorphism is the most problems facing maintainers and developers when they use an object-oriented technique. In this paper, we highlight an object oriented maintenance problem under software maintenance system. Furthermore, we discuss an OOMS as a new proposed solution for the Java programming language. Finally we discuss some relevant work.

REFERENCES

[1] Chow Paul K. O., and Yeung Daiel S. (1996) "Behavioral modeling in Object-Oriented Methodology" Information and Software Technology 38 (1996) 657-666.

[2] Lieberherr Karl J. and Xiao Cun (1993) "Object-Oriented Software Evolution" IEEE Transaction on Software Engineering, Vol. 19, No. 4, April 1993.

[3] Goyal, A. , Sharma, P. Goyal, S.B. and Singhal, N. (2012) "Analyzing Object Models with Theory of Innovative Solution" Second International Conference on Advanced Computing & Communication Technologies (ACCT), 2012 pp 46 - 50

[4] Liang Bao, Chao Yin; Weigang He; Jun Ge and Ping Chen (2010) "Extracting reusable services from legacy object-oriented systems" IEEE International Conference on Software Maintenance (ICSM), 2010 pp.1 - 5

[5] King, Jerry Gao, Pei Hsia, Yasufumi Toyoshima, Chris Chen, Young-Si Kim, and Young-Kee Song (1995) "Developing an Object-Oriented Software Testing and maintenance Environment" Communications of ACM October 1995/Vol. 38, No. 10.

[6] O'Hare A. B. And Troan (1994) "RE-Analyzer: From Source Code to Structured Analysis" IBM Systems Journal Vol. 33, No. 1, 1994

[7] Harrold Mary Jean and Mally Brian (1993) "A Unified Interprocedural Program Representation for a Maintenance Environment" IEEE Transactions on Software Engineering, Vol. 19, No. 6 June 1993.

[8] Fry, Z.P.; Shepherd, D.; Hill, E.; Pollock, L.; and Vijay-Shanker, K., (2008) "Analysing source code: looking for useful verb-direct object pairs in all the right places" Software, IET , Volume: 2, Issue: 1, pp.27 - 36

[9] Mayrhauser A. Van, and Vans A. M. (1996) "Identification of Dynamic Comprehension Processes During Large Scale Maintenance" IEEE Transaction on Software Engineering, Vol.22, No. 6, June 1996.

[10] Bennett Keith (1996) " Software Evolution: Past, Present and Future" Information and Software Technology 38(1996) 673-680.

[11] Hart Johnson M. (1995) "Experience with Logical Code Analysis in Software Maintenance" Software-Practice and Experience, Vol. 25(11), 1243-1262 (November 1995)

[12] Lee Byoung Y. and Lee Jee K. (1997) "A Knowledge-Based Maintenance of Legacy Systems: METASOFT" Expert Systems With Applications, Vol. 12, No. 4. Pp. 483-496, 1997.

[13] Babiker Elmamoun (1997) "A Model for Reengineering Legacy Expert Systems to Object-Oriented Architecture" Expert systems With Applications, Vol 12, No. 3 pp. 363-371, 1997.

[14] Wilde Norman and Huitt Ross "Maintenance Support for Object-Oriented Programs" IEEE Transaction on Software Engineering, Vol. 18, No. 12, December 1992.

[15] Jambor-Sadeghi Kamyar, Ketabchi A. Mohammad, Chue Junjie and Ghiassi M. (1994) "A Systematic Approach to Corrective Maintenance", The Computer Journal, Vol. 37, No. 9, 1994.

[16] Bellin Davis (1991) "Software Maintenance The small Systems management Guide" Prentice Hall, Englewod Cliffs, New Jersey.

[17] Landsaum Jerome B. And Glass Robert L. (1992) "Measuring & Motivating Maintenance Programmers" Prentice Hall Englewood.

[18] Huan Li; Beibei Huang; Jinhu Lu, (2008) "Dynamical evolution analysis of the object-oriented software systems ", (IEEE World Congress on Computational Intelligence). IEEE Congress on

Digital Object Identifier, pp. 3030 - 3035

[19] Zuylen H. J. Van, (1993), "The REDO Compendium, Reverse Engineering for Software Maintenance", West Sussex po17 IUD, England.

[20] Al-Fawareh Hamed, Abdul Azim Abd Gani, (1997), 20, May, " Software maintenance: State of the art ", Intec'97, Information Technology Colloquium, University Putra Malaysia.

[21] Al-Fawareh Hamed, Abdul Azim Abd Gani, (1997), "Reverse Engineering: Tools Comparison" REDECS'97, International Conference, University Sains Malaysia.

[22] Daconta Michael C. (1996) "Java for C/C++ Programmers" John Wiley & Sons, Inc. New York.