International Journal of Information Technology and Knowledge Management
July-December 2008, Volume 1, No. 2, pp. 485-495

WEB BASED CSM METHODOLOGY FOR OBJECT ORIENTED SOFTWARE SYSTEM

M. ZKurian & A. S. Manjunath

Abstract

This paper is on Comparative Software Maintenance for Object-Oriented software system. The Object-Oriented systems have the relationships such as Inheritance, polymorphism, encapsulation, information-hiding, aggregation and association combination, which can make the system complicated to maintain. The effects on the system caused by maintenance activity can ripple through system components complicating maintenance and testing of the system. A Comparative Software Maintenance locates potential side effects, ripple effects and other effects of maintenance. An improved impact analysis procedure that determines impact of changes to the component level is presented. The result of impact analysis is applied to determine the component level testing requirements. The CSM methodology is implemented in Web Based Development Environment using Java Web Server, Comparative Impact Analysis using Java an OO programming language and JFlex a software lexical analyzer for OO program maintenance.

Keywords: CSM, Impact Analysis, Ripple effect, JFlex, Web Based Development Environment OO System.

1. Introduction

Comparative Software Maintenance is a multi-stage maintenance methodology model that encompasses the following:

- Determines Object Oriented system components – classes, models and objects and interactions such as aggregation, inheritance and uses
- Models system components in the Enhanced Low-Level Software Architecture for the Object Oriented systems

Comparative Impact analysis is the task of identifying the potential consequences of a change, or estimating what needs to be modified to accomplish a change. Locating the effects of maintenance provides the maintainer with knowledge that assists in debugging and testing the affected components. The primary factor which poses a challenge for software maintenance is the problem of understanding the structure and relationships that exist between the components in the software. The knowledge required for program comprehension comes from many sources, including design documentation, personal experience, knowledge of the problem-domain, knowledge of components of

the programming language(s) used to write the system, Ahrens [1], and from observing the software in use.

This focuses on object-oriented (OO) systems. The structure and relationships in OO systems have the potential to represent exquisite designs and streamlined implementations. Often, though, OO software presents the maintainer with a jungle of interwoven and overlapping structure and interactions. Identifying the components that need to be modified is difficult. Determining the effect of maintenance activities on the system, a tedious and time-consuming task, represents a challenging problem for the maintainer.

Characteristics such as encapsulation help to centralize and organize the data objects. The data structures and code that are the data objects of the system are typically bundled. But OO software has other characteristics such as information hiding, inheritance, name polymorphism, aggregation and association, while providing versatility and extendibility make understanding the software a difficult task. Understanding the structure of the code and relationships among classes, methods and objects is key to the maintenance of object-oriented software. The actual modification of the code is relatively straightforward. Knowledge of what to change and of the components affected by the change is a difficult but crucial aspect of the maintenance process.

The OO characteristics that provide powerful capabilities for implementing software systems also introduce difficulties in program maintenance. These difficulties, as described by Kung *et al* [2], are summarized as follows:

Understanding problem is caused by encapsulation and information hiding. These characteristics cause a "delocalized plan" in which functions from several classes may be invoked to perform an operation. Functions from a called class may in turn call functions from other classes, resulting in a chain of calls. Invocation chains may become quite involved. The implication of the invocation chains is that a tester/maintainer has to understand sequences of member functions and the semantics of the class prior to preparing any test cases and/or modifying the intended functionality. Since it is necessary to understand all the parts in sufficient detail before testing/modification, this adds tremendous complexity for testing and maintenance of OO systems.

The Complex dependency problem is caused by the presence of inheritance, aggregation, association, template class instantiation, class nesting, dynamic object creation, member function invocation, polymorphism, and dynamic binding. These factors create complex relationships that cause dependencies between classes. The complex relationships contribute to:

(1) Difficulty in understanding a class in a large system.

(2) Difficulty in determining a starting point in testing either code modifications.

(3) The high costs of testing.

(4) Difficulty in discovering and testing.

(5) Difficulty in the identification of change impact in OO maintenance.

The tool support problem arises from a lack of OO based CASE tools. Program test sets are usually generated manually, consuming person-hours and most likely omitting certain affected code and execution paths. Therefore, tool support is crucial in the maintenance phase. Kung *et al* [2].

The objective of this study is to develop a methodology to improve the maintenance of OO software systems. The hypothesis of this is that the architecture of an object-oriented software system can be used to determine the impact of system changes.

2. BACKGROUND AND FOUNDATION

There are a numerous phases in the life of a software product. The waterfall model, as presented in Ghezzi *et al* [3], has five major phases. They are requirements analysis and specification, design and specification, coding and module testing, integration and system testing, and delivery and maintenance. This maintenance phase is the longest phase of the life cycle. Maintaining software becomes more difficult as time progresses and the system evolves.

The maintenance phase is a microcosm of the software lifecycle. Modifications to a software system require a thorough understanding of the system, integration of new requirements, development of new code, possible alterations to existing code structure, and testing. Problems often arise in the maintenance phase because of a lack of understanding of the true functionality and structure of the system. Original documentation is often incomplete or inaccurate due to previous modifications. Typically, the maintainer of the system was not a member of the original design team, but the maintainer must be as familiar with the system as the original design team. We now examine several important aspects of software maintenance.

Software maintenance is the modification of a software product after its delivery (to the customer) to correct errors, to improve product performance or other attributes, or to adapt the product to a modified environment IEEE STD 1219-1993[4]. These maintenance activities are categorized respectively as corrective, perfective and adaptive. Grubb [5] adds preventive as a type of maintenance activity. Corrective is concerned with finding and correcting faults that have been discovered by previous users of the software. This type of activity usually results in a “quick fix”. The problem is isolated and repaired quickly, and then the documentation and design are changed to reflect the modification Basili *et al* [6]. Perfective maintenance is any activity performed to make the software run faster, do more, or work better. Adaptive maintenance adjusts software

to function with new hardware or in a new environment. Preventive maintenance is undertaken to head off possible problems.

Impact analysis (IA) is the activity of identifying what to modify to accomplish a change or of identifying the potential consequences of a change, Arnold *et al* [7]. There are various approaches to IA including program slicing Poonawala etal [8], code analysis Kung *et al* [9], coupling measures Briand etal [10] (metrics) and tracing calls graphs and inheritance trees.

The ripple effect (RE) is the effect caused by making a small change to a system which affects many other parts of a system Arnold *et al* [11]. We define the ripple effect as a phenomenon that occurs when a change in one component in a program has an effect (typically unknown or unsuspected) on one or more other components. Ripple effect analysis (REA) is the recursive analysis of affected components emanating from the source of the change (both up and down) until all components involved are located and analyzed. REA is particularly important in object programming because of the nature of inheritance and other object characteristics.

3. COMPARATIVE SOFTWARE MAINTENANCE

Comparative Software Maintenance (CSM) is a methodology that models OO software relationships. It locates changes made to an OO software system as a result of maintenance, and predicts or determines the effects produced by the changes, facilitating testing of an OO system by locating affected components.

A Comparative Software Maintenance (CSM) methodology as in Fig. 1 has been developed to assist in addressing and understanding the complex dependency and tool

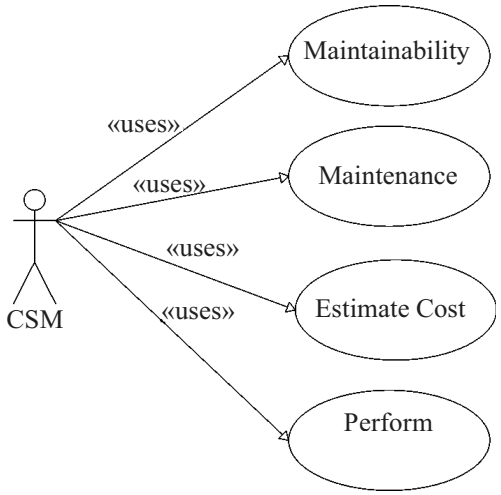

Figure 1

support problems. CSM is a process for aiding in program understanding and automating impact analysis to reduce testing effort arising from OO software maintenance activities. It provides the maintainer of OO software with detailed knowledge of the interactions and relationships among class, method and object components in an OO system.

CSM is a multi-stage maintenance methodology model that encompasses the following:

- Maintainability
- Maintainability Cost
- Estimation Cost
- Perform Maintenance

3.1. Web Based Development Environment

- It is a Development Environment, which provides features like creating any new project, new project folder, new project file, save, edit, find and replace, creating project related documentation. Figure 2 shows a three tier-architecture consisting of Web Page, Java Web Server consists of CSM Code using Java Servlet, Java and Database using Oracle.

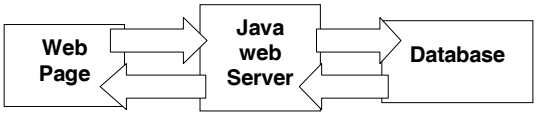

Figure 2

3.2. Enhanced-Low Level Software Architecture

Figure 3 shows an overview of ELLSA. ELLSA performs static and dynamic analysis of OO software systems. ELLSA provides a textual description of an OO system. ELLSA discovers relationship such as aggregation, inheritance, association and polymorphism, and other OO characteristics and display them in a textual description.

The ELLSA model serves as the basis for the Low Level Software Architecture model. We define textual description for the three components of the ELLSA – class, method and object. A textual description is a collection of descriptive data about a component in an OO system.

ELLSA of an OO system has two major phases. In the first phase the system code is parsed in order to locate and extract the obvious "raw" data and relationships such as class names, class method name, and parameter list. The second phase consists of the application of algorithms to the extracted "raw" data in order to discover the relationship

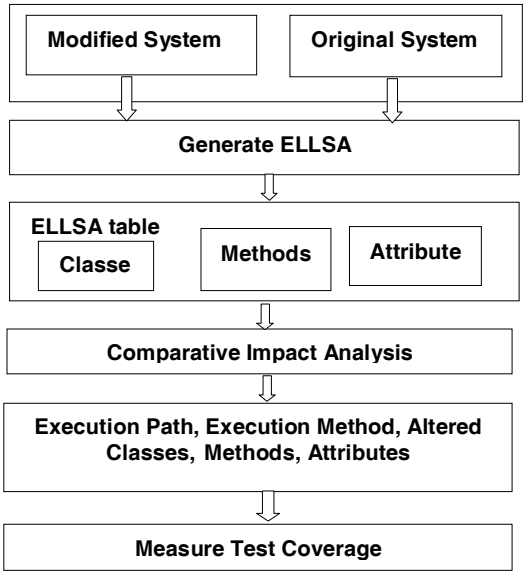

Figure 3

that are not obvious, such as descendent classes, used by, and called by. The feasibility of CIA is demonstrated by implementing in JFlex. The correctness of JFlex results is demonstrated by comparing results of the process with manually computed results.

3.3. Comparative Impact Analysis

- Determines OO system components – classes, methods and objects, and interactions such as aggregation, inheritance and uses.
- Models system components in the Enhanced- Low Level Software Architecture (ELLSA) for the OO system.

3.4. Lexical Analyzer

First phase of compiler is the Lexical Analyzer, also called as Scanner, which recognizes the basic language units called Tokens. JFlex implements the algorithms required to create the software architecture of a Java system perform impact analysis and generate test suites. JFlex demonstrates that the CIA process can be automated. The original source is not required as long as the ELLSA of the original system has been preserved.

4. CONTENT AND FORMALIZATION OF ELLSA

We define textual descriptions for the three components of the ELLSA – class, method and object. We describe the content of the ELLSA.

Informally, a textual description (TD) is a collection of descriptive data about a component in an OO system. A TD contains description of the class data members, listing of the class methods and description of the relationship and interactions a class has with other classes. For methods, TD contains description of the parameters and local variables of a method, object components contained in the method and interactions with other methods. For objects, TD contains data dependencies with other objects and usage description. TDs for all components enumerate the interfaces for the component. An interface is a description of how a component can interact with other components.

The class component TD duplicates some of the data found in both the object and method component descriptions. The duplication helps speed construction of the ELLSA. As an example of data duplication, a class needs to know the names of its methods. There may be dozens of methods listed for a class. The method, though, has only one class. Each method may create objects of some class. The textual description of the class declaring the method must list the classes created by all of its methods. This list results in duplication of the output, but it facilitates comprehension of the structure and interactions of the system.

5. THE CSM DATA MODEL

In order to store and represent the architectural clichés of an object oriented system and other program data, we define a data representation model. The data model is capable of capturing and storing information about individual internal class structure, external relationships with other classes, and the structure of method interfaces. Knowledge as local and parameter objects, call chains, uses and used by relationships, and information about the object instances of an object system must be recorded as well. The LLSA model serves as the basis for the Enhanced Low-Level Software Architecture (ELLSA) model.

6. EXTRACTION OF THE "RAW" DATA

CSM extracts raw data through two phases of ELLSA. The data extracted from a class declaration are:

- Class name
- Extends name or parent class
- Implements name
- Names, types and attributes of data members
- Names, return types, parameters and local variables of class methods
- Method invocations
- Assignment statement level data dependencies

7. CHANGES TO CLASS STRUCTURE

Classes are essentially data and methods. The maintainer can apply the following types of changes to classes

- add a data member
- delete a data member
- modify access attributes
- add a method member
- delete a method member
- change a method member

Some previous research has listed changing a data member's data type as a type of change that can be applied to a class. Data type "changes" are not possible. In our view, changing a data type effectively renames the data member. Changing the data type changes part of what defines a variable: the context in which it is allowed to operate legally. This is the same as deleting the variable and adding a new variable with the same name but different data type.

8. VISUAL IMPACT ANALYSIS

Gallagher [12] Employs program slicing to select a point in an ANSI C program for observation. The method looks at program variables and essentially models dependencies that exist among variables via assignment statements and parameter passing. The method is a visualization of the data collected by the Surgeon's Assistant and is called the Decomposition Slice Display System. A variable's decomposition slice is the set of all statements in the program that contribute to the variable's computation. According to Hutchins etal [13] Visual Impact Analysis has improved the recognition of further dependencies such as interference, a list of what other variables could "interfere" with the maintenance of some other variable.

9. IMPACT ANALYSIS SYSTEM (IAS)

IAS is a system for performing high-level impact analysis Barros *et al* [14]. IAS appears to be primarily concerned with function call graph dependencies. This approach is based on modeling of both the dependencies within the maintained software system and the way modifications induce ripple effects. The software system being maintained is modeled as a set of typed objects such as code modules, functions, design objects and test cases linked by various dependency links such as documentation, composition, and version links. These objects form classes (not in the OO sense) of relations such as Requirements, Validation-Test-Cases, Is-Tested-By, and Is-Composed-Of.

Changes to the software system also are modeled as types and links. Objects such as functions are modified and produce effects that are modeled by links such as is_called_by. Links are followed from object. IAS record "impacts" as they result from "propagation rules". A software tool implementing IAS was produced S.A.Bohner etal [15] that provides a high level view of proposed changes to a software system.

10. OOTME

Kung *et al* [16], has presented various types of code changes that occur with OO software systems. Changes are classified as data, class, method or class library changes. Rules, based mainly in graph and set theory, are presented for detecting changes in OO components and for detecting further affected components. Change impact identification is achieved through a prototype tool called the object-oriented test model environment (OOTME). OOTME implements the rules for change types and identification. A method for class firewall construction is also presented. A class firewall is the set of affected classes produced in an OO system whenever modifications are made to the system. OOTME provides a graphical representation of a OO systems that is similar to LLSA. The current main use of the OOTME is to design optimal test suites for OO software.

11. OMEGA

Chen *et al* [17], has presented an integrated environment for C++ program maintenance which describes three new dependence graphs specific to OO systems: message, class and declaration dependence in a model called C++ DG. Additional several new slicing techniques are presented. The use of the new dependencies and slicing on code maintenance is described, specifically as to the ripple effect and regression testing. The dependencies described, specifically as to the ripple effect and regression testing. The dependencies described are similar to the dependence problems described in Shrivastave *et al* [18]. The application of the discovered dependencies and program slicing leads to recursive analysis of the ripple effect caused by code modification. As the effects are located, classes and methods affected can be "marked" for testing or re-execution in the testing phase.

12. ALGORITHMIC ANALYSIS

Li *et al* [19] has presented four algorithms that measure the effect of proposed changes to OO systems. The ripple effect is calculated by application of algorithms that:

1. calculate the change effects inside of a class
2. calculate the change effects among clients
3. calculate the change effects among subclasses
4. measure the total effect by driving the algorithms in 1,2 and 3

Presents details of how different types of changes affect the system. Changes are broadly categorized as method or member change, and then refined to more detail such as adding a member or changing an attribute.

The algorithms calculate the transitive closure of each of the potentially effected classes and methods. With the added information that provides, it will be possible to greatly improve upon the information provided by the algorithms in Recognition of low-level design patterns, effects of data type changes, and effects of addition and deletion of classes can be drawn from the LLSA model of a OO system.

13. CONCLUSION

We have developed OO architectural studies found in OO systems. ELLSA algorithm compares and analyzes the impact on software components. It highlights the changes made in the software components such as classes, interfaces, methods and member variables. CSM carries the results of impact analysis and ripple effect analysis to the retesting of effected and changed components.

REFERENCES

[1] Transistion to a Legacy- and Reuse-Based Software Life Cycle, Ahrens, J., Prywes, N., *IEEE Computer* October 1995, **28** (10), (1995), 27–36.

[2] Developing an Object-Oriented Software Testing and Maintenance Environment, Kung, D., Gao, J., Hsia, P., Toyoshima, Y., Chen, C., Kim, Y., Song, Y., *Communications of the ACM*, **38**, (10), (October 1995), 75–87.

[3] Fundamentals of Software Engineering, Ghezzi, C., Jazayeri, M., Mandrioli, D., Prentice Hall Publishing, (1991).

[4] IEEE Standards, IEEE STD 1219-1993, *Software Maintenance standard*, IEEE Standards Office, (1993).

[5] *Software Maintenance*: Concepts and Practice, taking, A., Grubb, P., International Thomson Computer Press.

[6] Investigating Maintenance Processes in a framework-Based Environment, Basili, V., Lanubile, F., Shull, F., *Proceedings of the International Conference on Software Maintenance*, (1998), 256–264.

[7] Impact Analysis-Towards A Framework for Comparison, Arnold, R., Bohner, S., *Proceedings of the International Conference on Software Maintenance*, (1993), 292–301.

[8] Omega-an Integrated Environment for C++ Program Maintenance, Chen, X., Tsai, W., Hunag, H., Poonawala, M., Rayadurgam, S., Wang, Y., *Proceedings of the International Conference on Software Maintenance*, (1996), 114–123.

[9] Change Impact Identification in Object Oriented Software Maintenance, Kung, D., Gao, J., Hsia, P., Wen, F., toyoshima, Y., Chen, C., *Proceedings of the International Conference on Software Maintenance*, (1994), 202–211.

[10] Using Coupling Measurement for Impact Analysis in Object-Oriented Systems, Briand,, L., Wust, J., Lounis, H., *Proceedings of the International Conference on Software Maintenance*, (1999), 475–482.

[11] Impact Analysis–Towards A Framework for Comparison, Arnold, R., Bohner, S., *Proceedings of the International Conference on Software Maintenance*, (1993), 292-301.

[12] Improving Visual Impact Analysis, Hutchins, M., Gallagher, K., *Proceedings of the International Conference on Software Maintenance*, (1998), 294–301.

[13] Improving Visual Impact Analysis, Hutchins, M., Gallagher, K., *Proceedings of the International Conference on Software Maintenance*, (1998), 294–301.

[14] Supporting Impact Analysis: A Semi-Automated Technique and Associated Tool, Barros, S., Bodhuin, Th., Escudie, A., Queille, J. P., Voidrot, J. F., *Proceedings of the International Conference on Software Maintenance*, (1995), 42–51.

[15] S. A. Bohner. Software Change Impacts – An Evolving Perspective. *Proceedings of the International Conference on Software Maintenance,* (October 2002), 263–272.

[16] Change Impact Identification in Object Oriented Software Maintenance, Kung, D., Gao, J., Hsia, P., Wen, F., Toyoshima, Y., Chen, C., *Proceedings of the International Conference on Software Maintenance*, (1994), 202–211.

[17] Omega-an Intefrated Environment for C++ Program Maintenance, Chen, X., Tsai, W., Hunag,H.Poonawala, M.rayadurgam, S., Wang, Y., *Proceddings of the International Conference on Software Maintenance*, (1996), 114–123.

[18] Using Low-Level Software Architecture for Software Maintenance of Object-Oriented Systems, Shrivastave, C. and Carver, D., *Proceedings of 1995 Software Engineering Forum*, (1995), 31–40.

[19] Algorithmic Analysis of the Impact of Changes to Object-Oriented Software, Li, L., Offutt, A. J., *Proceedings of the International Conference on Software Maintenance*, (1996), 171–184.

M. ZKurian
Dr. MGR Educational and Research Institute
Chennai
E-mail: *mzkurianvc@yahoo.com*

A. S. Manjunath
Department of Computer Science and Engineering
Siddaganga Institute of Technology
Tumkur 572103
Karnataka, India