Maintenance of Object-Oriented Design

Subhash Singh Parihar[1], Kumar Saurabh[1]
[1]Department of Computer Science & Engineering,
Pranveer Singh Institute of Technology Kanpur, U.P., India
dr.s.s.parihar77@gmail.com
kumar.saurabh00@gmail.com

ABSTRACT— Maintenance is a set of activities to keep the system operational after the installation of software. These activities include residual bug fixing, software modifications and software extensions. The ever growing need for productivity resulted in finding new techniques for designing software such that the development and maintenance expenses are brought down and productivity is improved. Object orientation is one such technique that evolved. This paper is a step towards observing, whether developing a system using an object oriented design methodology is any beneficial over developing the same system using a functional approach during the maintenance phase of software development.

Keywords—Maintenance, Object, Design, Softwaremetrics, Object-Orientated Design

I. Introduction

The need to increase programmer's productivity, to develop and maintain large complex software in a competitive environment and the need for higher reliability of the developed system have driven interest in hunting for new approaches to software design and development. It is believed that software systems that are designed using Object oriented paradigm are likely to be easier to maintain than the systems designed using the standard function oriented approach.

Maintenance costs are a major problem in software development, for these costs over the life of the software. For exceed the initial software development costs. Maintenance comprises all the development related activities that are done after the software is delivered and installed. These are major reasons for the existence of maintenance work, resulting in three different types of maintenance. [7]

1. Corrective Maintenance –to fix bugs
2. Adaptive Maintenance –To adopt to changes
3. Enhancement Maintenance—to add more functions

There are two major problems in maintenance. First is how to make necessary changes to the software and the other is how to make sure that the changes have not caused other problems.

II. Object Oriented Design

Basic Concepts of Object Orientation

OO (Object-Orientation) has been announced to drive high-quality software and improve programmer effectiveness through code reuse. Here are some of the most common terminologies for object-oriented metrics [8]:

1. Object: An object is a kind of entity that can salvage a state and execute a variety of transactions on that state.
2. Class: A set of objects with a shared structure and a methodically expressed behavior. It serves as a model from which an element can be created.
3. Method: An object-based method available for all class instances does not need to be unique.
4. Inheritance: A connection between classes where one class inherits the features from one or more classes.
5. Cohesion: The degree to which the methods of one class are interrelated.
6. Coupling: refers to the level of direct knowledge that an Object A has of Object B.
7. Instantiation: Building an object instance and linking or adding data.

In an object oriented design, the system is decomposed based on objects and in contrast in a functional design the system will be decomposed based on the functionality. [5] The common elements of an object oriented design methodologies have evolved. The common elements of an object oriented design methodology are as follows: [3]

2.1 Methods of Object Oriented Design

- A set of methods to perform domain analysis called object oriented analysis.
- A notation (graphical and/or textual) to represent the function. Structures and control aspects of the system, identified at analysis stage.
- A set of methods to model the solution domain details, called object oriented design.

The first step in object oriented analysis is to identify the real world and conceptual objects that are specific to the problem domain that are called semantic objects, and then identify the attributes and behaviour of these objects.

Object oriented design step starts off with the end products from the analysis stage, and it refines them with solution domain details by enriching the objects with new features, or by restructuring or extending inheritance hierarchies or by adding new objects that are specific to solution domain like user interface objects or utility objects like stack, queue etc.

III. Maintainability of Object Oriented Software

In this paper we come to the issue regarding maintainability of object oriented software. The fundamental question is whether maintenance of object oriented software is easier than maintaining regular function oriented software [4]. As this question cannot be answered by formal analysis and proofs; we decided to explore its experiment ability. Experimental evaluating for answering this question is not easy. First, it requires a test bed of many problems where for each problem both object oriented software and regular software have been developed.

The Test bed

The test bed for the maintenance experiment we decided that we would provide the maintainers the software requirement specification and the code for the problems. No other information (e.g. about design, testing etc.) is supplied as this is frequently the situation in real life. For target languages we selected C and C++ as both are popular languages. As they are also similar in some sense, in syntax-by choosing these languages we tried to minimize the effect of programming language and such factors. We started off by choosing three pilot projects; whose C code was already developed using conventional approaches based on the available specifications of these programs (or by talking to their authors) the software requirement specifications were prepared. As a result, a set of documents comprising SRS in IEEE standard [2] form is delivered for all the three pilot projects. These SRS documents were reviewed with client for completeness and consistency. In the next phase of developing the test bed object oriented software was developed for all the three pilot projects. The object oriented design methodologies that are used to develop these pilot projects was based on Rumbaugh's approach [1]. C++ was used as the implementation language for the pilot projects developed using the object oriented design methodology. Later in the experiment we also ensured that the person who designed and implemented the software is not given the maintenance task.

A brief description of the three pilot projects comprising test bed is given below.

: Pilot Project#1:

This is a text formatting application that supports a subset of the functionality of the standard nroff utility this is shipped with many Linux flavours. It accepts a normal ASCII file, contain text to be formatted along with format specifications as input and produces the formatted text on standard output. Format specifications like spacing, paragraphing, listing, sectioning etc. are supported.

TABLE 3.1: PILOT PROJECT STATISTICS

Pilot Project#	SRS size (No. of words)	Size* of C code (LOC)	Size* of C++ code (LOC)
1	1133	511	1340
2	1970	1500	1900
3	1690	1345	1920

*: size of code includes any non-comment and non-blank lines

Pilot Project #2

This application generates a timetable of course schedules, based on some policy directions. It takes two text files as input one containing a list of valid room numbers and their capacities and a list of valid timeslots, and another file containing a list of preferences against each course that is to be scheduled.

Pilot Project#3

This application is a simulation of a disk file system, with flat file hierarchy. The application provides a user interface with options like format disk, create file, delete file, create record, delete record and search record etc.

: Observations

TABLE 3.2.1 FUNCTION MAPPING FROM C TO C++

Pilot Project#	Total no. of functions in C Program	No. of C Functions mapped onto C++ Program	No. of C function that could not get mapped onto C++ program
1	17	17	0
2	27	16	11
3	33	30	3

In C→ C++ direction

In table 3.2.1 some function could not get mapped onto C++ programs. In consisting whether a function of C program could be mapped onto C++ program or not we saw whether the functionality was at least similar to the functionality of same method of same object in C++ program through not exactly the same.

TABLE 3.2.2 METHOD MAPPING FROM C++ TO C

Pilot Project #	Total No. of methods in C++ Program	No. of C++ methods mapped onto functions in C Program	No. of C++ methods that could not get mapped onto functions in C program
1	87	54	33
2	88	46	42
3	90	58	32

In C++ → C direction

Method mapping from C++ to C direction showed some interesting results.

- First and foremost, reason is that objects were encapsulating data, so data could not be accessed directly in the methods. All access to data was through some standard interface in the form of set and get methods and such operations appeared in almost all classes. Whereas in C programs there were no such constraints so all such methods did not appear majority of the methods that did not appear in C Programs were of this sort.
- Some of the methods constructors and destructors for object. Even though some of the constructors did appear in c programs in the form of initialize functions in C programs. There was very less no. of initialize functions in C programs. Moreover, there were more than one constructor methods for some objects.
- Some object had some data members that did not appear in C program in one form or the other. So, naturally there were some methods operating on their data members which did not appear in C program.

While making the study we also observed the following

- In C Programs the binding between data and functions was loose, whereas in data and functions were bound more tightly. In C++ program any invocation of a

method of an object had to be qualified by the object name, unlike in C Programs where any function could be invoked freely since it was not bound to some data.

- Some objects had some data members that did not appear in C Program in one form or the other. So, naturally there were some methods operating on these data members which did not appear in C program.
- In C++ program constructors were used for initialization, but in a program there were explicit initialize functions for data structure.
- Export control is one place where there is difference between a C program and C++ Program. The access to data members is restricted by the export control (private, public or protected in C++). In C program there was no such information hiding.

IV. Modification Request

As a part of the maintenance experiment are collected some modifications requests from the clients for each pilot project. We also referred to them by requirement changes or just changes. [6]

Types of changes in the requirements.

i. Extends the functionality
ii. Adds new functionality
iii. Changes existing functionality
iv. Omits some existing functionalityv.

We tried to include all elements that are necessary to compare the maintenance effort between a non-object oriented versions and object oriented version. In the number of lines changed and added computation we did not include blank lines and comment lines. If a function is augmented without a single line being modified, then also we included it in the list of functions changed.

Any miscellaneous additions and changes not shown explicitly but are considered in total LOC changed/added computation.

To understand what all data has to be collected to correctly measure the effort required in making a change, one has to observe the sequence steps involved in making a change by a maintainer with SRS, source code and change description.

We used the following metrics to measure the effort in making change.

i. Total time taken to make change
ii. Change volume
iii. Add volume
iv. Change distribution

Total time taken to make change is consolidated time of reading and understanding the code, time spent on paper and pencil technique, time spent in editing, compiling, executing and debugging.

Change volume represents of the total number of lines changed as a part of required change. If some lines are deleted as a whole, then we counted them also in counting number of lines changed.

Add volume represents the total number of lines added as a part of required change.

Change distribution represents how a change is scattered along the program. This is a measure for how many functions or methods of the objects were affected by a change.

V. CONCLUSION

From the maintenance experiment we have observed that a system developed using object oriented techniques can accommodate changes during maintenance phase of the software development, more easily than the equivalent system developed using a functional approach. The total effort required to make a change thus will be considerably smaller proving object oriented design a cost effective technique during the maintenance phase.

An object oriented design brings a system closer to user's perspective of the system, making the person doing a change to perceive easily where to change

References

[1] Rumbaugh, J., M. Blaha, W. Premerlani, F. Eddy & W. Lorensen, Object-Oriented Model- ing and Design, Prentice Hall, 1991.
[2] Software Engineering Standards IEEE press, 2000.
[3] Booch, G., Object-Oriented Design, 2006.
[4] S. Henry, M. Humphrey and J. Lewis, "Evaluation of the maintainability of object-oriented software," IEEE TENCON'90: 1990 IEEE Region 10 Conference on Computer and Communication Systems. Conference Proceedings, 1990, .
[5] M. E. Fayad and D. C. Schmidt. Object-oriented application frameworks. Communications of the ACM, 40:32–38, October 1997.
[6] B. Lientz. Issues in software maintenance. ACM Computing Surveys, 15:271–278, 1983.
[7] S. R. Schach. Classical and Object-Oriented Software Engineering with UML and C++. WCB/McGraw-Hill, Fourth edition, 1999.
[8] International Journal of Science and Engineering Applications Volume 11-Issue 05, 66 - 71, 2022

eISSN 1303-5150