Software Size Evaluation Using Object Points

Thandar Zaw, Khine Khine Oo
University of Computer Studies, Yangon, Myanmar
thannarzaw@gmail.com

Abstract

Object-oriented technologies have emerged as a dominant software engineering practice. The growth of Object Oriented practices has required software developers and they have been estimating the size of their development projects. Several attempts have been made to categorize the complexities of software systems, and the complexity of software architectures is a subject of ongoing research. Software reuse limits the amount of new software that needs to be produced. Most method for estimation effort requires an estimate of the size of the software. The size of software is estimated by using complexity weight of object points. Object-oriented software is the process of systematically building the software systems using objects, types and classes. This paper presented the calculation of reuse object points in the modified software application by using object point estimation model. This size of the software code is estimated by using COCOMO-II object point analysis model. The normal size estimated after object point analysis is finalized to calculate the adjusted object points in reusing modified software application.

Keywords: Effort estimation, Object point analysis, COCOMO-II

1. Introduction

Software cost estimation is an important activity during its development. Cost has to be estimated continually during all the software development phases. Software cost estimation depends on the nature and characteristics of a project [9]. At any phase, the accuracy of the estimate depends on the amount of information known about the final product.

Software size estimation is mainly dependent on its size. Boehm [2] has developed Constructive Cost Model (COCOMO) to establish a relationship between the cost and size of software. The main disadvantages of using Lines of Code (LOC) as a unit of measure of software size are the lack of a universally accepted definition and language dependency.

For most companies trying to improve their software development performance, objects and software reuse must become key parts of their software engineering strategy. Succeeding with industrial-strength object-oriented software engineering requires that the promise of large-scale software reuse be realized in a practical way. [7]. Reuse efforts date back to 1994 when some divisions began to develop reusable components in Instrument Basic, Objective-C, and C++.

For the effective management of the resources in a project, it is necessary to estimate the function points of a system in the requirements, analysis and design phases. At the requirements phase the measurement is completely from users' perspective. Analysis phase is not only from an analyzers' perspective, but also includes users' perspective and at the design phase the measurement is completely from a designer's perspective.

In recent years, Object Oriented technology has emerged as a dominant software engineering practice. As it happens with many new technologies, the growth of Object Oriented practices has forced software developers and their managers to rethink the way they have been estimating the size of their development projects [5]. To measure OO software, the main components to be considered are raw functionality of the software, communication among objects and inheritance.

In this paper, a well defined estimation model in COCOMO-II is proposed which can be used to estimate the reuse object size required for developing the next software application. This paper provides the adjusted object point by estimating the sub-program (client) to refer the reference program (server).The model calculation is designed to help project manager to estimate effort at the very early stage of requirement analysis. For the size estimation of the program, the different techniques such as function point analysis, object point analysis and use case analysis are used.

The rest of the paper is organized as follows. In Section 2, we reviews related work. We show the general architecture of the proposed system in Section 3. Section 4 gives determine object points and a description of estimation of size. Section 5 describes experimental results of this system and section 6 draws some conclusion and future work.

2. Related Work and Background

As software grew in size and importance, it also grew in its complexity, making it very difficult to accurately predict the cost of the software development. The better accuracy in estimation can be achieved by developing the domain based useful models that constructively explain the development

life cycle and accurately predict the effort of developing the software. [8].

Over the years, different approaches have been suggested for the estimation of different types of projects. These projects are categorized as development projects, maintenance projects and support services by keeping in mind that these activities never occurred simultaneously. Most of these approaches estimate the effort based on the size of the development project and same approach is also used in estimating the effort of projects.

Teologlou [5] has proposed Predictive Object Points (POPs) based on the class hierarchy and weighted methods per class. From the class hierarchy, the counts of number of top level classes, average depth of inheritance tree and average number of children per base class are considered. Methods are weighted according to five types (Constructors, Destructors, Selectors, Modifiers and Iterators). At the design phase, the information about the data, aggregation and polymorphism is available but, this information is not considered by the POPs measure.

IFPUG [1] considered classes as logical files and methods as transactional functions from user's perspective during the analysis as well as design phases. This counting procedure lacks the ability to measure the inheritance and communication among objects.

Baresi et al. [3] investigated whether estimated effort provided by inexperienced developers can be used to estimate actual effort. It was quite clear that it is possible to use the estimated values as predictors for the actual ones; however other variables, such as size, also need to be incorporated to the model to make it more realistic and meaningful.

Reifer et al [10] proposed an extension to the COCOMO-II model by introducing WEBMO model. Reifer proposed different cost drivers by keeping in mind the demands of web projects. In WEBMO, size is measured by using analytical Halstead's formula for volume.

3. Proposed System Architecture

Different reuse attributes are visible when reuse is examined from different perspectives. In our system, we consider a system where individual modules access some set of existing software entities. A program unit (header File) reused is considered as a server and the unit accessing that program unit (Program File) is considered as client. Reuse can be observed from the perspectives of the server and the client. Each of these perspectives is relevant for the analysis and measurement of reuse in a system. A set of potentially measurable attributes can be derived for Object Oriented Programming Language based on profiles of reuse from each perspective. As a result, we can define measurement of object points that can be derived; they are presented in a set of complexity tables. Figure1 represent the proposed system architecture.

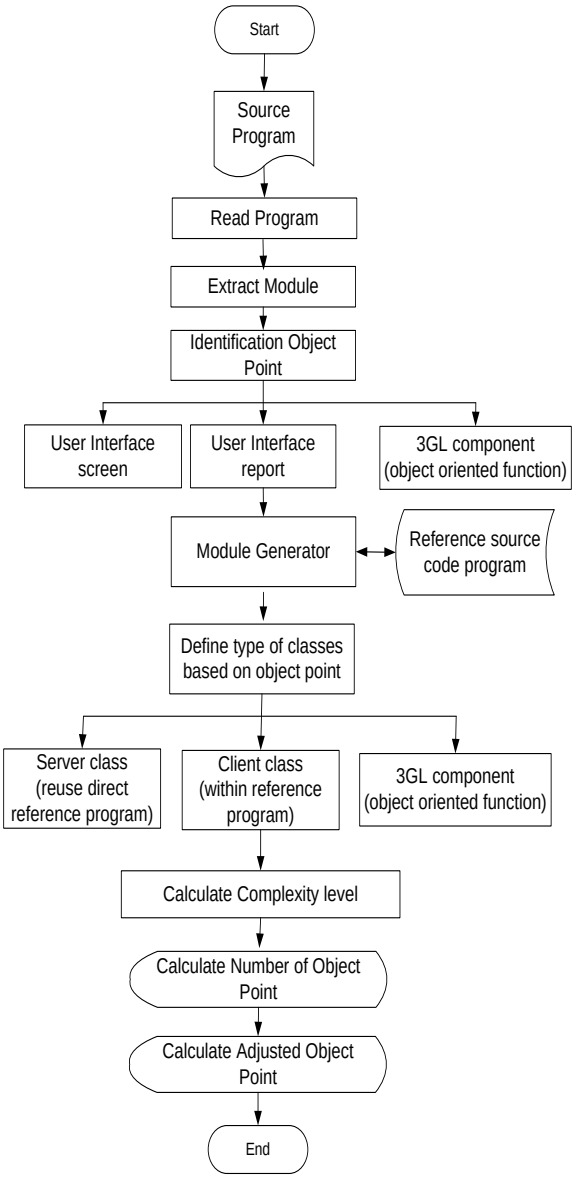

Figure.1. Proposed System Architecture

4. Estimation of Size

For Size calculation, we have opted for the Object point analysis. Object points are a measure of the size of computer application and the projects that build them. The size is measured from a components point of view. It is independent of the computer language, development methodology, technology or capability of the project team used to develop the application.

4.1 Identify System Boundaries

Identification of system boundary means limiting the scope of development. We can define what application is to be developed and what is outside the scope of development. This will help to find the application objects to be included in the system to count the total Object Points (OP).For each features of the system, we can easily determine the corresponding object points and their scope in the system.

4.2 Determine Object Points

The Object Points (OP) reflects the countable objectivity provided to the user by the application file/tool. Each tool has certain defined ways using which we can quantify these object points. The defined specific Objects should be evaluated in terms of what is being delivered. This means count only user defined and requested Objects. Do not include any of the existing objects and trivial objects. Object Point is the total of object points for all the objects delivered in any file. Identify the different objects from the business user's point of view. In each application, there are two types of objects, one which are directly visible through user interface and others which are not directly visible. The following different categories of object have been identified:

- **User Interface Screens**
  - Data Forms
  - Message Boxes
  - Error Messages
  - Interactions with Databases
  - No. of Java Script functions/ operations/Validations
  - No. of style classes uses
- **User Interface Reports**
  - Printed reports
  - Graphics Analysis Reports

4.3 Object Points in Object Oriented Design

In Object Points procedure, methods and data are separated while calculating the functionality of software. But a class encapsulates both data and methods. So, complexity of object oriented implies that both data and methods should be considered as a single entity. The functionality of the object oriented software is decided according to the data processed by the functions and communication among objects. From the user's perspective, we calculate class hierarchies, inherited data, and aggregation and method signatures.

4.4 Complexity of Each Object

All the objects are characterized by the attributes they posses and their behavior in different environments. The complexity if each object also depends upon how it is interlinked with other objects and number of instances it uses. Objects include screens, reports and modules in third generation programming languages. Object Points are not necessarily related to objects in Object Oriented Programming. The numbers of raw objects are estimated, the complexity of each object is estimated, and the weighted total (Object-Point count) is computed. The percentage of reuse and anticipated productivity are also estimated.

Table 1: Object Point Analysis-Screen

Number of views contained	Number and source of data tables		
	Total < 4 (<2 server, <2 client)	Total < 8 (2-3 server, 3-5 client)	Total 8+ (>3 server, >5 client)
< 3	Simple	Simple	Medium
3 – 7	Simple	Medium	Difficult
8+	Medium	Difficult	Difficult

Table 2: Object Point Analysis-Reports

Number of sections contained	Number and source of data tables		
	Total < 4 (<2 server, <2 client)	Total < 8 (2-3 server, 3-5 client)	Total 8+ (>3 server, >5 client)
< 2	Simple	Simple	Medium
2 or 3	Simple	Medium	Difficult
> 3	Medium	Difficult	Difficult

Table 1 and 2 represent the Object Points analysis for Screen and Reports respectively. In each table, the server means the number of data using in conjunction with the SCREEN or REPORT in header file and the client means the number of data using in conjunction with the SCREEN or REPORT in program files

4.5 Complexity Weight

As Table 1 and 2 in this section, design of the problem should be considered to estimate the complexity of each object. Then each object is classified into simple, medium and difficult complexity levels depending on values of

characteristic dimensions defined on the following table 3:

Table 3: Object Point Analysis-Complexity Weight

	Complexity		
Type of object	Simple	Medium	Difficult
Screen	1	2	3
Report	2	5	8
3GL component	N/A	N/A	10

4.6 Calculate Object Points

To the total object points in a particular feature or an application, we add up all the objects multiplied by their complexities in it. This gives us the total number of object points in the application. Each application involves certain redundant features. Further, object oriented development methodology inherently promotes code reuse. We consider code reuse as an important parameter while calculating total project points. Although Code reuse parameter vary from application to application and also depends upon the design of an application but still following factors may help us to determine this parameter:

- Number of views Screen
- Number of printed reports
- Number of function enhanced
- Amount of code used from previous application having same features

According to table 1, for example, we calculate one of program files has number of views screen (screen objects) is less than three if this object is less than four the total of header file and program file and less than two in these two files, is called as "Simple" type. Others type of objects is calculated in this same way. According to table 2, for example, we calculate one of program files has number of sections (report functions) is less than two if this object is less than four the total of header file and program file and less than two in these two files, is called as "Simple" type. Others type of objects is calculated in this same way.

To evaluate complexity weight, if the type of screen objects is "Simple" type, we multiply one with these objects. If the type of report objects is "Simple" type, we also multiply two with these objects. Especially, the type of 3GL components has only "Difficult" type; we multiply ten with these 3GL objects.

Finally, the effect of reuse% is taken the percentage of screens, reports, and 3GL components reused from header file.

%reuse= (used components/total components)/100 …(1)

The inputs are the NOP (Number of Object Points) of reference file. The following formula is used to estimate adjusted object points in our subprogram.

Adjusted OP=NOP* (1-reuse%/100) …(2)

5. Experimental Result

In this experiment, data is collected from the two different types of file. Reference file contains all screen , report and 3GL components function , is called reference file(header file). Client files contain some function reuse the reference file, is called class file. This paper tested 3class file refer to the same referenc file. Different class files have vary in the complexity weight in their implementation. To demonstrate the measurement, we designed and implemented the prototype C++ language. Firstly, we estimate the number of screens, reports and 3Gl components from the header file that will comprise in this program file.Second, we classify each object instance into simple, medium and difficult complexity levels depending on vales of characteristic dimensions using Table 1 and 2. Third, we weight the number in each cell using the Table 3. The weights reflect the relative effort required to implement an instance of that complexity level. Fouth, we determine object points that add all the weighted object instances to get one number, the object-point count.Fifth, we estimate the percentage of resue you expect to be achieved in next program file, and then we will compute the adjusted object point to be developed in each program file.

Client 1:
2 screens: 1 Simple+2 Difficult
60 3GL components: 60 functions

1 Simple screen *1 =1
2 Difficult screen *3 =6
60 3GL component *10 =600
NOP(number of Object Point)=607
%reuse= 2.970
Adjusted Object Point(AOP)=588.97

Client 2:
1 screens: 1 Simple
22 3GL components: 22 functions

1 Simple screen *1 =1
22 3GL component *10=220
NOP(number of Object Point)=221
%reuse= 10
Adjusted Object Point(AOP)=198.9

Client 3:
1 screens: 1 Simple
2 report: 2 Simple
31 3GL components: 31 functions

1 Simple screen *1 =1
2 Simple report *2 =4
31 3GL component *10=310
NOP(number of Object Point)=315
%reuse= 6.0606
Adjusted Object Point(AOP)=295.909

6. Conclusion

An organization can also use the reuse measures to monitor the success of a reuse program in promoting reuse in development. The measures can quantify precisely how much percentage of reuse is taking place. Developers can use the reuse data and representations to produce customized project to satisfy specific goals. In this paper, object points defined screens, reports and 3GL components in testing program files. This paper calculates the number of object points by analyzing the complexity of weights for all objects. The percentage of reuse estimated in header file and program files you expect to be achieved in next program file, and then we computed the adjusted object point to be developed in each program file. Reuse measurement will also help users to develop new software that is easily reused. This system will use object point estimating technique for cost effort approach. Currently, this paper developed for measuring reuse components in programs implemented in C#.

References

[1] B. W. Boehm. "Software Engineering Economics",PrenticeHall, Inc., Englewood Cliffs, New Jersey, USA, 1981.

[3] Baresi, Li, Morasa, S., & Paolini, P., "An empirical study in the design effort for web applications", Proceedings of Web Information System Engineering, 2002.

[4] D. Janaka Ram and S.V.G.K. Raju, "Object Oriented Design Function Point", Distributed and Object Systems Lab, Indian Institute of Technology Madras, IEEE, 2000.

[5] G. Teologlou, "Measuring object oriented software with predictive object points", In 10th Conference on European Software Confrol and Metrics, May 1999. Available at http://www.escom.co.uk/publications.

[6] IFPUG. Function Point Counting Practices: Case Studies Release 1.0 - Case Study 3 - Object Oriented Analysis, Object Oriented Design. International Function Point Users Group, Westerville, OH, 1996.

[7] I.Jacobson, M.Griss and P. Jonsson,"Software reuse: Architecture, process and organization for business success",1997.

[8] N. Aggarwal, N. Prakash and S. Sofat, "Content Management System Effort Estimation Model based on Object Point Analysis", International Journal of Computer Science and Engineering, 2008.

[9] P. Jalote. "An Integrated Approach to Software Engineering". Narosa Publishing House, New Delhi, India, 1998.

[10]Reifer, D.J., "Web development: Estimating quick-to-market software", IEEE Software, Nov-Dec 2000.

[11] S.Axwl, "Position Paper: Towards Complexity Levels of Object Systems Used in Software Engineering Education",