Object Oriented Design Function Points

D. Janaki Ram and S. V. G. K. Raju
Distributed & Object Systems Lab
Indian Institute of Technology Madras
Chennai
India - 600 036
{janaki, raju}@lotus.iitm.ernet.in
http://lotus.iitm.ac.in

Abstract

Estimating different characteristics viz., size, cost, etc. of software during different phases of software development is required to manage the resources effectively. Function points measure can be used as an input to estimate these characteristics of software. The Traditional Function Point Counting Procedure (TFPCP) can not be used to measure the functionality of an Object Oriented (OO) system. This paper suggests a counting procedure to measure the functionality of an OO system during the design phase from a designers' perspective. It is adapted from the TFPCP. The main aim of this paper is to use all the available information during the OO design phase to estimate Object Oriented Design Function Points (OODFP). The novel feature of this approach is that it considers all the basic concepts of OO systems such as inheritance, aggregation, association and polymorphism.

Keywords: *Object Oriented design, Function point analysis.*

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 [7]. At any phase, the accuracy of the estimate depends on the amount of information known about the final product.

Software cost estimation is mainly dependent on its size. Boehm [3] has developed COnstructive COst MOdel (COCOMO) to establish a relationship between the cost and size of a software. The main disadvantages of using Lines of Code (LOC) as a unit of measure for software size are the lack of a universally accepted definition and language dependency [10], [11], [12].

TFPCP estimates the functionality of a software from a users' point of view at the requirements phase of the software development process [1], [2], [6]. It is based on the complexity of logical files and transactional functions derived from the users' requirements. A logical file is a group of logically related data and a transactional function provides the functionality to the user for processing the data by an application [6]. TFPCP should change its perspective as it is applied at various phases of software development. Also, TFPCP objective. However, function points may not overcome the disadvantages of LOC completely, it is more useful than LOC measure [7], [2], [9].

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 analysers' perspective, but also includes users' perspective and at the design phase the measurement is completely from a designer's perspective.

In recent years, OO technology has emerged as a dominant software engineering practice. As it happens with many new technologies, the growth of OO practices has forced software developers and their managers to rethink the way they have been estimating the size of their development projects [20]. The TFPCP [6] is not suitable for measuring the functionality of OO software. To measure OO software, the main components to be considered are raw functionality of the software, communication among objects and inheritance [20].

In this paper, a method for estimating the function points of OO systems at the design phase from a designer's perspective is developed. The OODFP counting procedure is adapted from TFPCP. It is estimated by two factors. One is the functionality of the OO software and the other is the complexity of each class, which depends on the design of the problem.

The rest of the paper is organized as follows. In section 2, the existing literature on OOFP is explained. Section 3 explains, why class complexity depends on the problem domain. The counting procedure of OODFP is explained in section 4. In section 5, the OOFP and OODFP are compared and discussed. In the last section, the scope for future research and the conclusions are given.

2. Related Works

Teologlou [20] 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 [5] 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.

Whitmire [21] considered each class as a logical file and methods sent across the application boundary as transactional functions. Schooneveldt [19] has treated classes as logical files and services delivered as transactional functions. These methods are similar to IFPUG counting procedure.

2.1. Caldiera's Counting Procedure

Caldiera [4] has developed a tool to count OOFP at various phases of OO software development. It is based on the adaptation of TFPCP to OO software. He also considered classes as logical files and methods as transactional functions.

Logical files are considered from four different perspectives viz., Single Class, Aggregations, Generalization/Specialization and Mixed. The weight of each logical file depends on the Data Element Types (DETs) and Record Element Types (RETs) it has. The data inherited from a base class is not considered to estimate the weight of a logical file. The communication among objects is also considered as DET/RET for a logical file. A single valued association is considered as a DET. A multivalued association is considered as a RET, because an entire group of references to objects is maintained in one attribute.

Transactional functions are weighted according to their signatures. Abstract methods are not counted and concrete methods are counted only once in the class in which they are declared.

The inherited data from a base class might be used by its derived classes. So, from the designer's perspective inherited data should be counted to estimate the complexity of a derived class. Also, method signatures are available at the design phase. So, the complexity of communicating objects should be considered for a particular method.

Though the above said counting procedures defined class as a logical file they considered only data within that class. This is not correct. Since defining class, as a logical file must includes both data and methods.

3. Complexity of a class

In OO systems, it is possible to have different classes with same number of DETs, RETs and/or FTRs (File Reference Type) but varying complexity. It is hard to design an OO system in such a way that all the derived classes use all the inherited data. Similarly, it is possible that some of the derived classes might not provide any functionality for the abstract methods. This information is available at the design phase according to the design of the problem. So, the design of the problem should be considered as a factor for estimating the complexity of each class.

The OOFP counting procedures in the literature has not considered the complexity of a class as a factor to estimate the functionality of a system.

4. Object Oriented Design Function Points

This section explains the OODFP counting procedure at the design phase from a designer's perspective. This procedure is adapted from TFPCP.

In TFPCP, methods and data are separated while calculating the functionality of software. But, a class encapsulates both data and methods. So, complexity of OO implies that both data and methods should be considered as a single entity. The functionality of the OO software is decided according to the data processed by the functions and communication among objects.

The design phase contains more information than the analysis phase such as class hierarchies, inherited data, aggregation, and method signatures. The OOFP by Caldiera [4] is not suitable at OO design phase, since it is calculated from the user's perspective. So, OOFP counting procedure should be modified at the design phase.

4.1. Counting Procedure

At the design phase, a logical file is a collection of data elements which are *visible* to all methods of a class. Transactional functions are the methods in a class.

Data in a class can be mapped to a logical file and each method to a transactional function.

4.1.1 Logical File

A logical file is divided into two types depending on the application boundary. Logical file within the application boundary is known as Internal Logical File (ILF) and outside the application boundary is known as External Interface File (EIF).

The complexity of an ILF/EIF depends on the DETs and RETs it has. A DET is a simple data type such as int, char, etc. Object reference, a complex data type is considered as a RET. So, a RET provides the complexity due to aggregation.

The inherited data is visible to all the methods in a derived class. So, inherited data should be included to calculate the complexity of a derived class. After DETs and RETs are counted, IFPUG [6] tables can be used to classify the logical files as low, average or high.

4.1.2 Transactional Function

Object models rarely contain the information needed to tell whether a method performs an input, an output or is dealing with an enquiry [4]. So, no distinction is made among the methods. A method in a class operates on the data within that class, arguments and return values. Since, the data within a class is already considered as a logical file, it can be omitted while calculating the complexity of a method. So, the complexity of a method depends on the DETs and FTRs in its signature. FTRs are analogous to RETs.

The inherited methods will be coded only once in the base class. So, methods that are inherited from a base class should not be considered for estimating the complexity of a derived class. If any derived class overrides a method, its complexity should be considered for that derived class alone.

An abstract method will be declared in a base class and defined in its derived classes. The complexity of an abstract method should be considered only for derived classes.

Using the signature of a method, it is possible to identify the communicating objects. So, association should be considered for the method from where it invokes the required method(s). A single valued association is considered as a DET and a multivalued association is considered as a FTR.

If a method does not have any arguments and return type, then its complexity is considered as one DET.

The method complexity is rated according to TFPCP's external input rate complexity table. To translate complexity rate to unadjusted function points external output conversion table is considered. So, this gives the worst case complexity.

4.1.3 Complexity of a Class

As explained in the section 3, design of the problem should be considered to estimate the complexity of each class. The complexity of a class is classified as low, average or high. The guidelines to classify the complexity of a class are given in Table 1.

Table 1. Class Complexity Classification

low	if a class processes less than 50% of data that is visible to it
average	if a class processes 51% to 70% of data that is visible to it
high	if a class processes more than 70% of data that is visible to it

The values of each low, average, and high are mapped to a numerical value based on observation across different projects. These values are presented in Table 2. It is named as Class Complexity Value (CCV).

Table 2. Class Complexity Value

	CCV
low	0.3
average	0.6
high	0.9

4.1.4 Unadjusted Function Points

Calculate Unadjusted OODFP (UOODFP) of the OO system as described here:

1. Calculate the function points for each class in the design. It is obtained by adding the function points of its logical file and transactional function.
2. From the Tables 1 and 2 estimate CCV.
3. UOODFP of a class is obtained by multiplying its function points with CCV.
4. Add UOODFP of each class to the get the UOODFP of the OO system.

4.2. Example

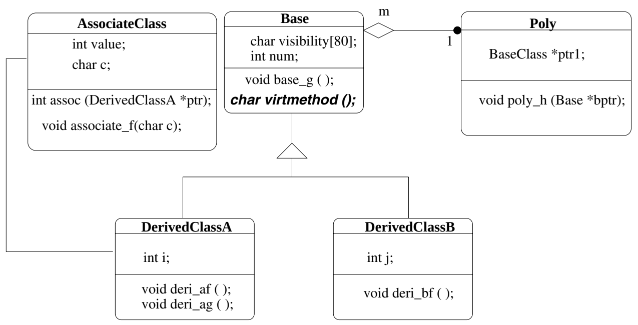

Figure 1. A simple example to measure the OODFP

OODFP is calculated for an example shown in Figure 1. This figure is drawn according to Object Modeling Technique (OMT) [17] notation. The method *virtmethod* in the class *Base* is an abstract method. The DETs and RETs/FTRs of each logical file and method are tabulated in the Table 3.

Table 3. DETs and RETs/FTRs of the design shown in Figure 1

Logical File/ Transactional Functions	DETs	RETs/ FTRs
Base	2	1
PolyClass	0	2
AssociateClass	2	1
DerivedClassA	3	1
DerivedClassB	3	1
base_g	1	0
virtmethod	2	0
poly_h	0	2
assoc	1	1
associate_f	1	0
deri_af	1	0
deri_ag	1	0
deri_bf	1	0

Poly class has two RETs (because of the *Base* class reference and the class itself). *DerviedClassA* has three DETs (two due to the inherited data from *Base* class, and one due to the data element it has). The method *poly_h* has 2 FTRs (one due to the *Base* class reference and other due to the multivalued association with *Base*). The signature of the method *poly_h* has a reference to the *Base* class and the *Base* class has a virtual method, So, there is a multivalued association with *Base* class. Similarly, *assoc* has one DET (due to the single valued association with *DerivedClassA* and one RET because of the *DerivedClassA* reference).

The given example is not specific to any problem, so CCV is not considered for calculating UOODFP.

The total UOODFP estimated for the OO design as shown in Figure 1 is 59 FPs.

Adjusted OODFP can be estimated by using the Value Adjustment Factor (VAF) as specified in the IFPUG document [6].

In this section, it was explained how the basic concepts of OO should be taken into consideration while calculating the OODFP. Also, the definitions of logical file and transactional function were defined from designer's perspective. It was explained how to consider the design of the problem as a factor for estimating function points of each class.

Finally, the developed OODFP counting procedure considered the following features of OO Systems at the design phase.

- Complexity of OO software implies that both data and methods should be considered as a single entity. This is achieved by counting the functionality according to the data processed by the functions and communicating objects.
- Inheritance is considered by including the inherited data to estimate the complexity of derived classes. But, inherited method is not counted, as it will be coded only once in the base class.
- Aggregation is considered by defining an object reference as a RET.
- Association is considered according to the communicating objects.
- Polymorphism is considered by defining multivalued association as an FTR.
- Complexity of a class which depends on the design of the problem is also considered.

5. Comparison

OODFP counting procedure is applied on different projects [18], [16], [14]. Schmidt [18] has developed application level gateways using framework components and design patterns. The gateway systems were implemented in C++ with the Adaptive Communication Environment (ACE) software. Simulation of a mechanism for achieving dynamic protocol switching in a multi protocol network using JAVA RMI has been developed by [16]. Different projects across different domains have been developed during software engineering course [14]. OOFP and OODFP are calculated at the design phase for the above said projects. The calculated OOFP and OODFP is converted to LOC based on [8]. The LOC based on OOFP and OODFP are given in the Tables 4, 5 and 6.

Table 4. Caldiera's OOFP Counting Procedure

Project	OOFP	Estimated LOC
ACE	198	5940
Protocol Switching	43	1290
Bank Simulator Project	308	9240
Railway Reservation	170	5100
Traffic Simulation	256	7680
Library Management	210	6300
Online Examination	132	3960
Weather Monetering Simulation	83	2940

While calculating OOFP for the above said projects using Caldiera's method the following point has been observed:

- All logical files and transactional functions have low complexity. But, the actual code varied considerably for each class. This is because of not considering the class complexity according to the design of the problem and improper calculation of function points at the design phase.

From the Tables 4, 5 and 6 it is evident that OODFP counting procedure gives more accurate value than the Caldiera's method for estimating the function points at the design phase.

6. Conclusions

A method for estimating the OODFP by adapting TF-PCP is presented. This method gives a new approach to estimate the OODFP from designer's perspective at the OO design. New definitions have been defined for a logical file and transactional function at the OO design phase. OODFP uses all the available information at the design level and covers all the basic concepts of OO systems. OODFP is useful to estimate cost of the project more accurately. So that it is possible to organize the resources in a more efficient way for a particular project.

Table 5. OODFP Counting Procedure

Project	OODFP	Estimated LOC
ACE	121	3630
Protocol Switching	23	690
Bank Simulator Project	110	3300
Railway Reservation	68	2040
Traffic Simulation	86	2580
Library Management	77	2310
Online Examination	68	2040
Weather Monetering Simulation	27	810

OODFP counting procedure can be applied on sufficiently large projects across different domains to validate the class complexity value. OODFP counting procedure can be adapted to Pattern Oriented Technique (POT) [15], [13] to estimate the functionality of a pattern.

References

[1] A. J. Albrecht. Measuring application development productivity. In *Proc. of the Joint SHARE/GUIDE/IBM Applications Development Symposium*, October 1979.

[2] A. J. Albrecht and J. E. Gaffney. Software function, source lines of code, and development effort prediction: A software science validation. *IEEE Transactions of Software Engineering*, SE-9(6):639–648, November 1983.

[3] B. W. Boehm. *Software Engineering Economics*. Prentice-Hall, Inc., Englewood Cliffs, New Jersey, USA, 1981.

[4] G. Caldiera, G. Antoniol, R. Fiutem, and C. Lokan. Definition and experimental evaluation of function points for object-oriented systems. In *Proc. of the* 5^{th} *International Symposium on Software Metrics*, pages 167–178, November 1998.

[5] 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.

[6] IFPUG. *Function Point Counting Practices Manual Release 4.1*. International Function Point Users Group, Westerville, Ohio, 1999.

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

[8] C. Jones. Backfiring: Converting lines of code to function points. *IEEE Computer*, 29(11):86–87, November 1995.

Table 6. Actual Lines of Code

Project	Actual LOC
ACE	3841
Protocol Switching	538
Bank Simulator Project	3450
Railway Reservation	1908
Traffic Simulation	1976
Library Management	2523
Online Examination	2831
Weather Monetering Simulation	632

[9] C. F. Kemerer. Reliability of function points measurement: A field of experiment. *Communications of the ACM*, 36(2):85–97, February 1993.

[10] A. V. Levitin. How to measure software size, and how not to. In 10th *International Computer Software and Applications Conference*, pages 314–318, Chicago, 1986.

[11] G. C. Low and D. R. Jeffery. Function points in the estimation and evaluation of the software process. *IEEE Transactions on Software Engineering*, 16(1):64–71, January 1990.

[12] J. E. Matson, B. E. Barrett, and J. M. Mellichamp. Software development cost estimation using function points. *IEEE Transactions on Software Engineering*, 20(4):275–287, April 1994.

[13] D. J. Ram. *Pattern Oriented Technique (POT)*. Available at http://lotus.iitm.ac.in/pot.html.

[14] D. J. Ram. Software engineering assignments. Available at http://lotus.iitm.ac.in/~se.

[15] D. J. Ram, K. N. Anantharaman, K. N. Guruprasad, M. Sreekanth, S. V. G. K. Raju, and A. A. Rao. An approach for pattern oriented software development based on a design handbook. *Annals of Software Engineering*, 10, 2000. To appear in Annals of Software Engineering, Vol. 10, 2000.

[16] D. J. Ram, R. I. Rajith, and M. Umamahesh. An implementation for protocol switching. Technical report, Indian Institute of Technology Madras, Chennai, 1999. Available at http://lotus.iitm.ac.in/~techrep.

[17] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. *Object-Oriented Modeling and Design*. Prentice-Hall of India Private Limited, New Delhi, India, 1998.

[18] D. C. Schmidt. *Implementation of Application Level Gateways using ACE*. Available at http://www.cs.wustl.edu/~schmidt.

[19] M. Schooneveldt. Measuring the size of object oriented systems. In R. Jeffery, editor, *Proc. of the* 2nd *Australian Conference on Software Metrics*. University of New South Wales, Sydney, Australia, 1995.

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

[21] S. A. Whitmire. *Applying Function Points to Object Oriented Software*. Software Engineering Productivity Handbook, McGraw-Hill, 1993.