International Journal of Modern Engineering Research (IJMER)
www.ijmer.com Vol.2, Issue.4, July-Aug 2012 pp-1923-1928 ISSN: 2249-6645

Oofp: Mapping the Oose Models into Function Points: Rules, Tool and Case Study

[1]Durga Gehlot, [2]Prof. Meena Sharma
[1]*Departmen of Computer Engineering, Institute of Engineering and Technology, DAVV Indore, India*
[2]*HOD of Computer Engineering, Institute of Engineering and Technology, DAVV Indore, India*

Abstract: *Function point analysis is useful to measure size of software projects in terms of functionality requested by user. The main advantage of function point analysis is that it is independent of the technology used for implementation. When we apply function points to object-oriented software projects, the concepts of development method have to be mapped into abstract models that contain functional items of the application. This proposed idea implement a tool for mapping function points into use case driven OOSE (object-oriented software engineering) Jacobson approach. In this idea we only considers analysis phase of OOSE life cycle. OOFP tool measures function point from requirements models and analysis model.*

Keyword: *function points, size, requirements model, and analysis model, OOFP.*

I. INTRODUCTION

Function Points Analysis (FPA) is one of the earliest models that are used to predict the size of software in the early stages. Albrecht proposed the FPA model in 1979 and it measures the size of software based on its functionalities [1]. The main advantages of the FPA model are that it is independent of the technology. Up to the present, various FPA versions based on the Albrecht's version have been proposed (e.g. IFPUG method, MarkII, COSMIC-FFP and they have been accepted as ISO/IEC standards. The current version of counting rules is recorded in the Counting Practices Manual [2]. This counting method isimplicitly based on the high- level model of software applications. Though independent of implementation, counting rules are thus based on the implicit assumptions on the abstract model of software applications. The items in the abstract model that are than counted include transaction and file types. These items are typically identified from documents of traditional, structured design technique e.g. data flow diagrams, hierarchical process models or database structures. The proposed paper focus on object oriented models based on the OOSE Jacobson approach. The transaction and file type items are counted from models of analysis phase. The models of analysis phase of OOSE includes use case model, domain object model and analysis model but this research paper focuses on counting function points from analysis model and use case model.

1.1Function Point Analysis with object – oriented design methods

The Function Point software measure does not require a particular development technique. However, the high level concepts of object-oriented development methods cannot be mapped directly to the concepts of Function Point Analysis. In order to apply this software measure early in the development process, the object-oriented concepts corresponding to transactional and data function types have to be determined.

Object-oriented methods differ, especially in their early development phases. The Object-Oriented method of Jacobson et al. is based on so-called use cases. The OO-Jacobson identifies the functionality of an application with requirements use case model. Data types are described with domain or analysis object model on the requirements level. The work proposes rules to map these models into function point counting procedures. With proposed rules, it is possible to count software developed with the OO-Jacobson method.

In this research paper, we focus on the approach of Jacobson et al [3]. This method is called Object- Oriented Software Engineering. The OOSE method defines a process to transform formalized requirements into a sequence of models. The steps include the requirements, analysis, design, implementation and testing models. The use case model is the basis on which all the models are developed. Together with the domain object model it forms requirements model as shown in Figure 1.

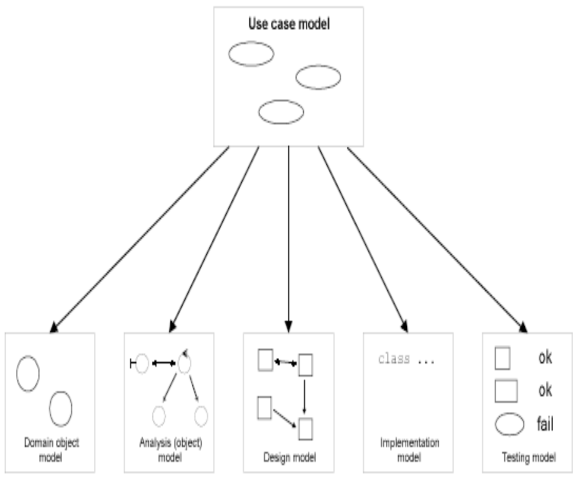


Figure 1: The use case model is the basis on which all other models of OOSE approach are developed.

The objectives of this research paper are:

1. The application of Function Point Analysis following the IFPUG standards.
2. To measure for software developed with OOSE method.
3. To count early in the life cycle, in the requirements analysis phase.

1.2 Related Work

Little work has been published on Function Point Analysis in the context of object-oriented software engineering techniques. But these approaches are based on a model that consists with objects together with their methods. In these approaches objects are treated as data files and methods as transactions which are the counting items in the Function Point Analysis. These approaches do not applicable to early OOSE documents. It is also questionable whether each individual method is to be counted as a transaction.

Whitmire[4] considers each class as an internal logical file and treats messages sent outside the system boundary as transactions.

The ASMA paper takes a similar approach. Services delivered by objects to the client are considered as transactions. The complexity of services is weighted based on accessed attributes and communications. Objects are treated as files, their attributes determining their complexity.

IFPUG [5] is working on a case study which illustrates the use of the counting practices for object-oriented analysis and design. This case study, which is currently in draft form, uses object models in which the methods of classes are identical with the services recorded in the requirements. Under this assumption, the methods can be counted as transactions.

Karner [6] proposes a new measure called Use Case Points for projects developed with the OOSE method. The structure of this measure is similar to Function Points, but it does not conform to the concepts of Function Points.

Thomas Fetcke[7] proposes rules for mapping the OO-Jacobson approach into Function Point Analysis. This paper is based on Thomas proposed rules. This research paper considers how to apply these rules to OOSE models to measure data and transaction functions. In this paper data files and transactionfunctions counting is done using models of analysis phase of the OOSE life cycle. Analysis phase model includes use case model and analysis model.

II. Brief Introduction To Oose

The OOSE method is divided into three major consecutive processes: analysis, constructive and testing. The analysis phase is further divided into two steps called requirements analysis and robustness analysis as shown in Figure 2. The first step derives the requirements model from the informal customer requirements. This model is expressed in terms of use case model, and may be augmented by a domain object model. The second step, robustness analysis, then structures the use case model into the analysis model by applying use case analysis. The succeeding process furthertransforms these models, as indicated in Figure 1.

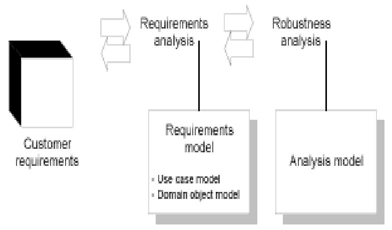


Figure 2: Analysis Phase of the OOSE life cycle.

At the focus of our work are the models developed in analysis phase as shown in Figure2. As Jacobson et al. state, the requirements model can be regarded as formulating the functional requirements specification based on the needs of the users. The goal of this research paper work is to count Function Points early in the life cycle, measuring the functionality requested by the user from these models. The overview of these three models discussed in [7].

III. Fuction Point Concepts

3.1 Function Point model

A high level model of the FPA mode is given in Figure 3 [7]. The function Point model specifies which component types of the software application will be measures and from which viewpoint this will be done. Hat is to be counted, and measured, are the internal files and external files of the application, together with the inputs, outputs and inquiries from and to the user. Software components or deliverables which are not visible from a user viewpoint are not considered part of the Function Point measurement model.

Function Point Model

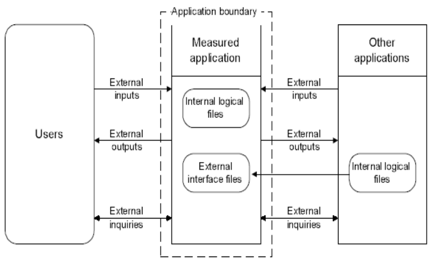


Figure 3: High-level view of the abstract Function Point model with users and links to other applications. The dotted line marks the application boundary.

3.2 Function Point Counting Process

In the IFPUG version, the counting procedure of function point consists of the following seven steps. The details of these seven steps are discussed in IFPUG [2].

1. Determine type of function point count.
2. Identify the application boundary (A boundary indicates the border between theapplication or project being measured and the external applications or the user domain. A boundary establishes which functions are included in the function point count).
3. Identify and rate transactional function types to determine their contribution to the unadjusted function point count.
4. Identify and rate data function types to determine their contribution to the unadjusted function point count.

5. Determine the Unadjusted function point counts.
6. Determine the value adjustment factor (VAF) that takes so-called global system characteristics into account, e.g. data communication, performance or end user efficiency. This adjustment is external of and independent from the concepts of the abstract FPA model. The global system characteristics determine an adjustment factor that is multiplied with the unadjusted count.
7. Calculate the adjusted function point count.

The next section describes the proposed mapping of OOSE models to function points along these five steps.

IV. Mapping Concepts (Proposed Method)

The aim of this research paper is to calculate the Unadjusted Function Point. The paper work proposes the following five steps to apply IFPUG version to the OOSE requirements analysis models (use case model and analysis model).

4.1 Step1(Determine the type of function count): This paper handles only the application project function point.

4.2 Step2(Identify the application boundary): The counting boundary is determined by the type of actors appeared in use case model of OOSE requirement analysis phase.

Proposed mapping rules to identify the application boundary

1. Accept each human actor as a *user* of the system.
2. Accept each non-human actor, which is separate system not design to provide functionality solely to the system under consideration as an *external application*.
3. Reject each non-human actor, which is part of the underlying system, e.g. a rational database system or a printing device.

The result is a representation of the application boundary as a set of users and applications external to the one under consideration as shown in Figure 4.

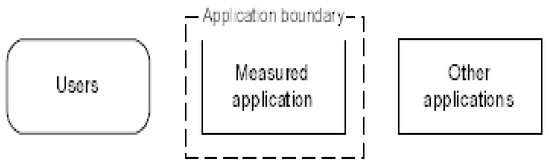


Figure 4: Step2- Identification of the counting boundary.

4.3 Step3 (Identify and rate transactional function types to determine their contribution to the unadjusted Function point count.):

The transaction functions are automatically decided based on actors and use cases of the use case model. Use cases are the OOSE concept corresponding to transactions.

Proposed mapping rules to identify and rate transaction function types

4. Select every use case that has a direct relation to an actor accepted by rule 1 or 2. This use case will be a candidate for one or several transactions.

5. Select every use case that extends a use case selected by rule 4 as a candidate.

6. No other use cases will be counted.

Determining the types of transaction (external input (EI), external output (EO) and external inquiry (EQ) is based on a set of detailed rules in FPA [1]. The rules are recorded in the IFPUG Counting Practices Manual [2]. The relevant sections are:

"External Input Counting Rules",
"External Output Counting Rules", and
"External Inquiry Counting Rules".

The rates of transactions are based on detailed rules in the counting practices Manual. The rules require the determination of data element types (DET) and file types that are referenced (FTR), illustrated in Figure 5

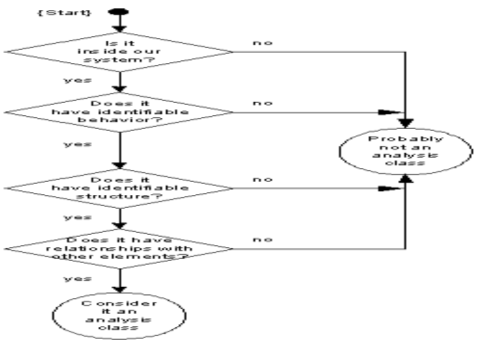
.

4.4 Step4 (Identify and rate data function types to determine their contribution to the unadjusted function point count.):

Data files are automatically decided based on analysis classes of analysis model. In the analysis model, the objects are typed into three groups, namely entity, interface and control objects. The set of objects that have to be analyzed is limited to the entity objects and is thus set of objects typically smaller. Interface (boundary) and control objects are part of implementation.

4.4.1 Finding of Analysis Classes from Use Case Model

1. Is this candidate inside our system boundary? **2.** If not, it might be an actor of our system. Does this candidate have identifiable behavior for our problem domain? (i.e., can we name the services/functions that are needed in our problem domain and that this candidate would own and provide?)
3. Does this candidate have identifiable structure? (i.e., can we identify some set of data this candidate should own and manage?)
4. Does this candidate have relationships with any other candidates?


{Start}
Is it inside our system?
no
yes
Does it have identifiable behavior?
no
yes
Does it have identifiable structure?
no
yes
Does it have relationships with other elements?
no
yes
Probably not an analysis class
Consider it an analysis class

If you find a "no," then the candidate is probably not a class; move on to the next candidate. If the answer is "yes," keep asking the questions. If you get all "yes" answers, conclude the candidate is a class, and get the next candidate to evaluate.

Proposed mapping rules to identify and rate data function types

7. Select every object of entity type as a candidate for a logical file.
8. No other objects will be selected.

For aggregation relationships

9. An entity objects that is part of another object (is aggregated into another object) is not a candidate for a logical file, but it is a candidate for a record element type (RET) for the file related to the aggregating top-level object

For inheritance relationships

10. An abstract object is not a candidate for a logical file. It is a candidate for a RET for each object that inherits its properties.

11. Sub-objects of a concrete object are candidates for a logical file or for a RET of that object.

12. If use cases make implicit use of logical files that are not represented in the analysis object (class) model, these files have to be included in the set of files.

Determining the types of data files (internal logical files (ILF) and external interface files (EIF)) is based on a set of detailed rules in FPA [1]. The rules are recorded in the IFPUG Counting Practices Manual [2]. The relevant section is “ILF/EIF counting rules”.

The rates of data files are based on detailed rules in the counting practices Manual. The rules require the determination of data element types (DET) and record element types (RET), illustrated in Figure 5.

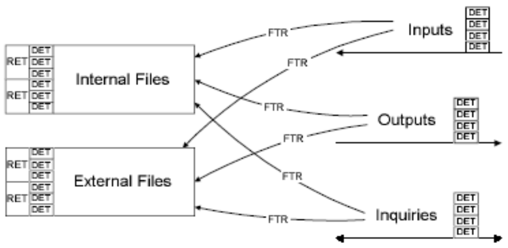


Figure 5: Step3 and Step4- Rating of data and transaction function types.

Proposed Mapping Rules for determining DET, RET and FTR

13. Attributes of objects are candidates for data element types (DET) for files and for the transactions by which it is read and maintained.

14. Candidates for record element types are determined by subgroups of files and by rules 9-11.

15. Each object maintained and /or read by a use case counts as a file type referenced (FTR) for the associated transaction(s), if and only if the object has been identified as a file in step 4.

After determining DETs, RETs, FTRs rate the data files and transaction functions according to rating matrix and then allot weighs according to weighting matrix in the Counting Practices Manual [2]. The total weight of data files types and transaction types are the required unadjusted function points count.

4.5 Step 5 (Determine the unadjusted function point counts)

As the result of Step3 and Step4, the counts for each function type are automatically classified according to complexity and then weighted. The total for all function types is the unadjusted function point count.

V. CASE STUDY

In this section the rules proposed in section 4 are applied to OOSE approach based project. The documentation provided included use case models and analysis object models together with the textual description these models.

5.1 Example of OOSE based analysis models

In OOSE life cycle, analysis phase is divided into two phases: requirement analysis and robustness analysis. Requirement analysis consists with two models, use case models and domain object models. Robustness analysis consists with a model known as analysis model. This research work focuses on use case models and analysis classes.

Case Study: A part of Course Registration System Use Case Model

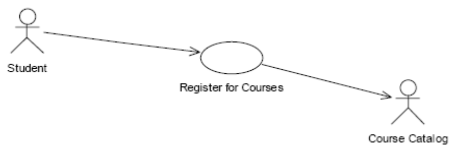


Figure6: Use Case: Register for Courses

This use case allows a Student to register for course offerings in the current semester. The Student can also update or delete course selections if changes are made within add/drop period at the beginning of the semester. The Course Catalog System provides a list of all the course offerings for the current semester.

5.2 Finding Analysis classes from use case model (Behavior)

Register for Courses Use Case-This use case starts when a Student wishes to register for course offerings, or to change his/her existing course schedule.

1. The system requests that the Student specify the function he/she would like to perform (Create a Schedule, Update a Schedule, or Delete a Schedule).
2. Once the Student provides the requested information, one of the sub flows is executed. If the Registrar selected “Create a Schedule”, the Create a Schedule sub flow is executed. If the Registrar selected “Update a Schedule”, the Update a Schedule sub flow is executed. If the Registrar selected “Delete a Schedule”, the Delete a Schedule sub flow is executed.

Candidates for entity (noun) and applying 4 conditions of section 4.4.1 to make them entity analysis classes as shown in Figure7.

1. Student-A person enrolled in classes at the university.

2. Schedule-The courses a student has selected for the current semester.

3. Course Offering-A specific delivery of the course for a specific semester – you could run the same course in parallel sessions in the semester. Includes the days of the week and times it is offered.

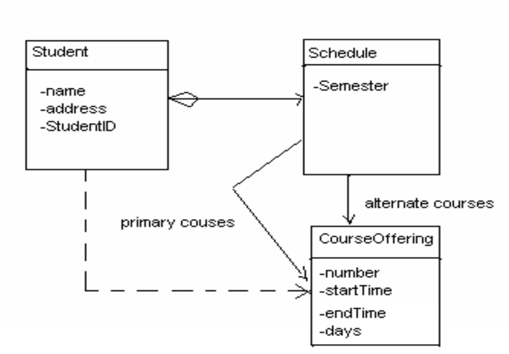


Figure7: Analysis class model consists with entity analysis classes

5.3 Now, applying proposed rules to Analysis Class Model and Use Case Model of above Case Study:

By rules 1, 2 and 3 application boundary includes-
Human actors- student
Non-human actors-Nil

By rules 4-12 and 13-15 candidates for transaction/data function types with ratings and weight are shown in Table1.

Transaction/ data function name	Transaction/ data function type	No. of DET	No. of RET/ FTR	Complexity	Weight/ Value
Register for courses	EI	3	2FTRs (Student, Schedule)	low	3
Register for courses	EO	3	2FTRs (Student, Schedule)	low	3
Register for courses	EQ	4	1FTR (CourseOffering)	low	4
Student	ILF	3	1RET(Schedule)	low	7
Course Offering	ILF	4	1RET(Schedule)	low	7
UAFP					24 FPs

Table1: Transaction/Data function types with ratings and weight.

In order to make it adjusted function point, we have to calculate and tabulate the GSC and come out with the VAF as shown in Table2.

GSC	Value (0-5)
Data communications	1
Distributed data processing	1
Performance	4
Heavily used configuration	0
Transaction rate	1
On-Line data entry	0
End-user efficiency	4
On-Line update	0
Complex processing	0
Reusability	3
Installation ease	4
Operational ease	4
Multiple sites	0
Facilitate change	0
Total	**22**

Table2: Global System Characteristics (GSC)

So using formulae:
VAF = 0.65 + ((sum of all GSC factor)/100).
= 0.65 + (22/100) = 0.87

This factor affects the whole FP like anything, be very particular with this factor.
So now, calculating the Adjusted FP (AFP) = VAF * Total Unadjusted FP (UAFP)
= 0.87* 24 =20.88,
=Rounded to 21 FPs

Compared with the approaches proposed in the literature, these mapping rules have certain advantages.

1. The mapping rules are based on the standard FPA defined in the IFPUG Counting Practice Manual. This widely used measure independent of technology.
2. The count is based on requirements models, which are the first models available in the OOSE life cycle. For the purpose of effort estimation based on Function Points, this is an essential prerequisite.

This approach also has some limitations.

1. The main limitation is the focus on the Jacobson OOSE method. Mapping rules are based on the requirements models of this approach and cannot be applied to methods that do not develop these models. Also an advantage this focus on OOSE, that the models are unambiguously defined in the method.

VI. OOFP TOOL

The concept of mapping the object oriented software models into function Points lead to implement a tool called OOFP. This tool is implemented in java language. The inputs for the tool are use case model and analysis class (object) model and the output includes the values of function points, transactional functions, and data functions.

The OOFP tool is automated using XMI (xml metadata interchange) parser. The XMI parser takes .xmi or .xml files of use case and analysis class models as input then read and extracts the use cases, actors, entity classes from these files. Extracted candidates are then used for calculating function points.

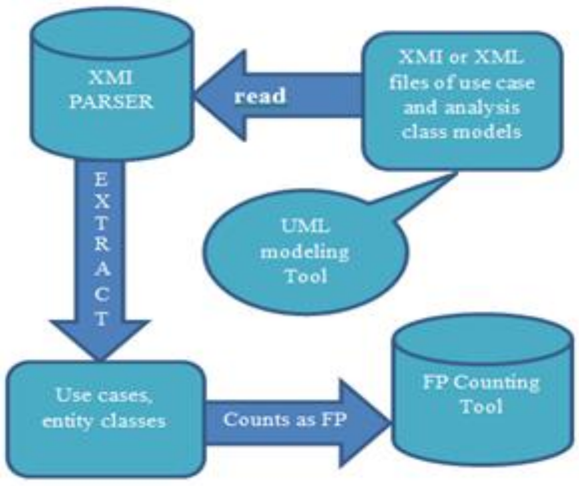

XMI PARSER
read
XMI or XML files of use case and analysis class models
UML modeling Tool
EXTRACT
Use cases, entity classes
Counts as FP
FP Counting Tool

Figure: 8 Function Point Counting Tool (OOFP tool).

VII. SUMMARY

This work demonstrated the applicability of function points as a measure of functional software size to the object- oriented Jacobson approach, OOSE. This work supports that the function point measures independent of the technology used for implementation and that it can be used in the object-oriented paradigm.

Future work in the field has to deal with the mapping of OOSE design model into function points.

References

[1] A. J. Albrecht. Measuring Application Development Productivity. IBM Applications Development Symposium, Monterey, CA, 1979

[2] Function Point Counting Practices Manual, Release 4.0. Westerville. Ohio, International Function Point Users Group, 1994.

[3] Jacobson, I., M. Christerson et al. Object-Oriented Software Engineering. A Use Case driven Approach. Addison-Wesely, 1992.

[4] Whitmire, S.A. Applying function points to object-oriented software models, 1992. In Software engineering productivity handbook. J. Keyes, McGraw-Hill, pp.229-244.

[5] Function Point Counting Practices: case Study3- Object-Oriented Analysis. Object-Oriented Analysis. Object-Oriented Design (Draft), International Function Point Users Group, 1995.

[6] Karner, G. Resource Estimation for Objectory Projects, Objectory Systems, 1993.

[7] ThomasFetcke, Alain Abran and THo- Hau Nguyen.Mapping the OO-Jacobson Approach into Function Point Analysis, 1997.Published in the Proceeding of TOOLS- 23'97.

[8] "CostEstimatingWebSite", http://cost.jsc.nasa.gov/COCOMO.html

[9] "Construx Estimate tool", www.construx.com

[10] "CoStar tool", www.softstarsystems.com

[11] "An introduction (tutorial) to Function Point Analysis", www.devdaily.com/FunctionPoints.

VIII. APPENDIX: Screenshots For OOFP Tool

The following screenshots show how function points are calculated using OOFP (Object oriented function point) tool.

Screen1: Function Point Calculation

Screen 2: Value Adjustment Factor

Screen 3: Function Point Report for a Project