A Sizing Approach for OO-environments

John Kammelar, CFPA
IQUIP Informatica B.V.
P/O Box 263
1110 AG Diemen
Netherlands
E-mail: j.m.kammelar@iquip.nl

Abstract

FPA is by far the most popular high quality sizing method for a traditional development environment. It complies to certain degree to the ISO standards for a 'Functional sizing method' (FSM) [12]. When FPA is applied to Object Oriented development methods the OO concepts and characteristics have to be translated into FPA terms. As a result the outcome of the function point count is difficult to relate to effort estimation. But the need for a seamlessly fitting Functional sizing method (FSM) for OO-environments is growing fast now in the emerging environment of Component Based Development (CBD).This document proposes a new FPA-alike estimation technique for OO-environments in such a way that the determined functional size is composed from elements which may be candidates for reusable software components. This approach is not considered a final product but rather as a starting point for further elaboration to develop an estimation approach for CBD.

Introduction

OO-concepts

According to the OO-concepts and characteristics objects manifest themselves to the user through their data and behavior. Both data (attributes) and behavior (operations) are encapsulated in autonomous components. Objects are classified into groups (classes) and communicate with other objects or with the user by means of passing messages. User requirements are fulfilled through services provided by the system. A service performs one or more elementary functions; a group of services which achieves a specific goal for the user is considered a use case. One or more use cases may comprise a workflow; a business process encompasses one or more workflows. The context (functional domain) of our elaboration consists of business information systems

Why FPA does not fit into OO

The usual approach for applying FPA to an OO-environment can be summarized as 'mapping the OO-concepts into the FPA abstract model and follow the existing FPA rules'. This approach results in the misfits as described hereafter. We characterize this approach as 'paradigm translation'. In figure 1 below, this approach is illustrated.

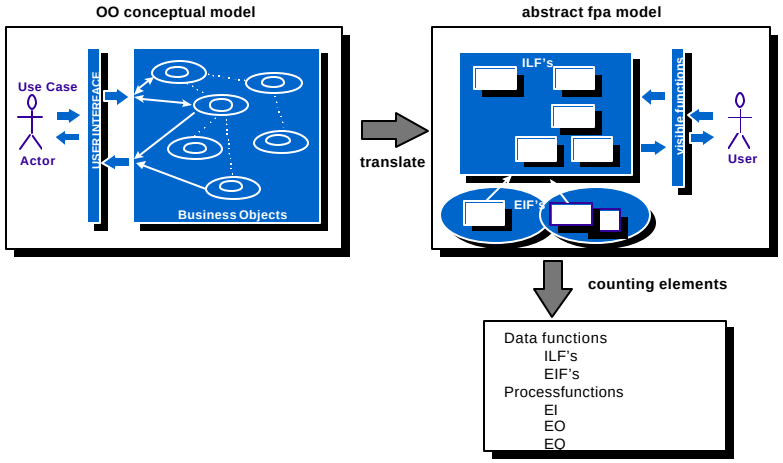


fig. 1 Paradigm translation

This approach is described as a set of mapping rules by Thomas Fetcke et al. [1].

The following shortcomings of FPA in relation to an OO-environment are encountered.

a. The separation of a function point count into a number of 'data-related' points and 'process-related' points is contradictory to the OO-paradigm.

b. The proportional assignment of function points to 'process' and 'data' is questionable and does not relate to OO-concepts.

c. Over-representation of visible functionality (data elements).

d. Assignment of functionality to class(es) is not provided.

e. The functional break down directly into FPA-functions (elementary process) fail to recognize (reusable) components.

These shortcomings impose that FPA needs to be modernized in a way that it can cope with new development environments and that it becomes sensitive for reusable functionality.

The new approach, which can be described as 'applying the line of thought of FPA to the OO-conceptual model' has never successfully been elaborated. We characterize this approach as a 'paradigm shift'. This approach is illustrated by figure 2.

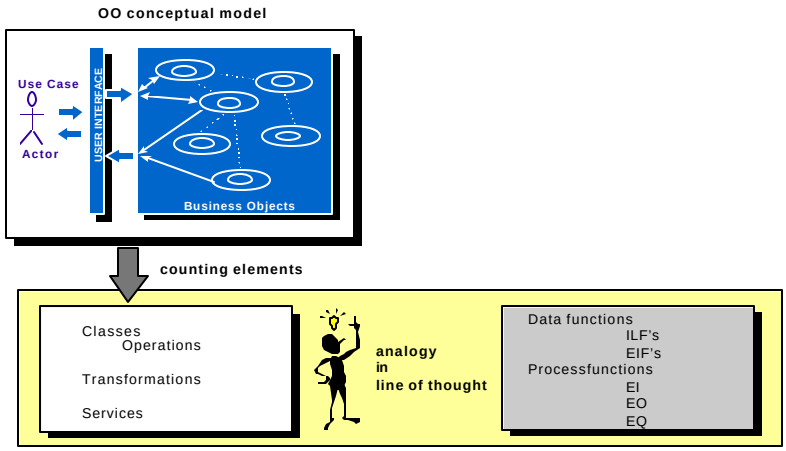


fig. 2 Paradigm shift

The line of thought of FPA is applied to the OO-conceptual model and encapsulated elements. New counting rules and definitions are provided for new counting elements. This approach results in a new FPA-alike sizing method which fully comply with mainstream OO-terms.

Objectives of a new approach

The objective of this approach is to express the functional size of a OO-application in terms of its development context. Therefore the functional user requirements (FUR's) are mapped to OO-functional components. It is emphasized that the proposed counting types are implementation technique independent. The functional size is expressed in *component object points(COP's)* to avoid confusion with function points. By means of COP's one is able to measure the functional size of object-oriented or object-based components. With respect to the emerging CBD-technology, the sizing rules accumulate elementary object/component-sizes to higher level components. It is an experimental FPA-alike FSM for OO-environments that does not suffer from the addressed shortcomings of FPA.

The focus is on functional sizing for purpose of project estimation and management. We have experienced that for these purposes functional size measurement which does not take into account the development context is insufficient accurate. With this respect there is a relation between the purpose and the scope of the measurement. The relevant purpose/scope combinations are indicated in the table below.

Purpose	**Scope**	**Required Accuracy**	**Metaphor**
Portfolio Analysis	*Organization*, Business Process, Entire information requirement (separated in domains)	*Low* (Development-Environment *in*dependent)	Artist impression of a building, total volume (m^3) and type of volume (storage, office, etc.)
Project Estimation, Tender comparison Budgeting	*Application* Entire information-requirement (separated in domains	*Medium* (Development-Environment dependent)	Architect drawing, specification, environment/neighborhood
Project Management Scheduling Task assignment	*Component* Information-requirements in their context	*High* (Implementation-Technique dependent)	Workplan and actual constructing together with the materials and purchased parts

Purpose/Scope combinations for size measurement

The described approach focuses on Project Estimation in an OO-environment, whereas FPA is perfectly applicable to Portfolio Analysis and Project Estimation (in a traditional development environment). Approaches using OO-metrics primarily focus on project management [3, 5, 9, 11] and quality [2, 6].

Related Work

Functional sizing and effort prediction in an OO-environment have been subject for elaboration in the last decade. The majority of these studies deal with metrics/measures and focus rather on quality aspects than on size and related development effort. Chidamber & Kemerer [2] are recognized as founders of useful metrics for OO design. More recently they described the managerial use of

metrics for object oriented software [3] showing quantitative and significant insight into the impact of OO design decisions on managerial variables like cost and productivity. Lorenz & Kidd [4] have elaborated a set of meaningful metrics for measuring project progress and quality. The purpose of their effort reads: "helping real development teams on OO projects estimate, schedule, and measure quality more effectively." Haynes, Menzies and Philips [5] attempted to use classes and methods as the basis for early effort estimation. They have measured productivity figures at the class level. Ramaskrishnan [6] describes related work. With respect to our approach the findings of Hastings [7] are of importance. He studied the applicability of FPA tot contemporary systems and concluded that FPA and derivatives do address all the needs of contemporary systems, in particular the ability to adequately measure complexity. But it is not clear whether he recognizes FPA (applied to OO environments) as inadequate from the scientific and engineering perspective.

Primarily focused on size estimation are the four steps sizing approach of Laranjeira [8]. Lower level class sizes are aggregated into higher level classes and finally the system size equals the sum of the sizes of the top level classes. Zhao & Stockman [9] have extended the Larangeira model with physical size factors and reuse size factors. This approach determines the size with a higher degree of detail. A correlation between FPA and the number of objects and methods was found by Catherwood, Sood & Armour [10], which may be understood as a valuable contribution in the field of FPA and effort prediction for OO-systems. The most recent remarkable approach is the one developed by PRICE systems and described by Teologlou [11] as size measurement for OO software using *predictive object points* (POP's). This approach uses well-known OO-metrics for purpose of effort prediction. Coincidentally this development took place during the same time frame as the one described in this report.

Approach

Conceptual sizing model

The figure below shows a conceptual sizing model for OO-environments. The white boxes show the entities which determine the functional size of an application. The entities exist within the gray rectangles representing the *User Domain* and the *System Domain* separated by the *user interface*. This separation of concern represents the border between the "what" and the "how" aspect. The columns at the sides show specifications, recommended in OO-literature. Model and diagram names refer to UML [13, 21]. The Definition of Elementary Process is an exception. This is a FPA ground rule and is used to assure the same degree of granularity between the different sizing methods.

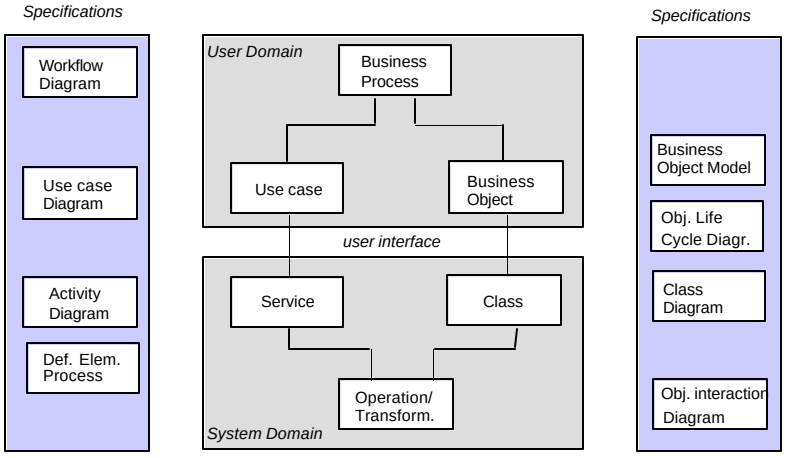


fig. 3 Conceptual sizing model for OO-environments

To our opinion the model is generic and as such a sizing-paradigm. The user domain comprises the elements by which the size of a process can be determined fully independent of the development/implementation environment. The elements in the user domain are expressed as FUR's and define the system in implementation-technique independent terms.

The system domain comprises elements by which the FUR's are implemented. The type of elements may vary according to the chosen development environment. In an OO-environment the computational support for a use case will be implemented by services provided by objects, grouped in classes. Services may comprise a number of operations carried out by (a collaboration of) classes. Classes, structures and operations are the elements to which the functional size is assigned: the Base Functional Components (BFC's).

The conceptual model is in line with the positioning and perspectives of use case ("outside the system") and collaboration ("inside the system boundary") of UML [21]. This approach describes how to measure the functional size of the computational support of use cases in OO-terms. It provides two types of count each with their own degree of accuracy to be used for purpose of project estimation and project management .

Counting elements

User Domain Elements (FUR's)

A business process is a coherent complex of business activities which aim for a discrete goal and for which a computer application have to be developed. The focus is on the process as a whole. The activities within a business process are modeled as use cases.

The use case has the proportion of a series of activities. The formal definition which is used within the scope of this report reads: a sequence of activities (usually determined by unity of time, place and action) which is carried out under the responsibility of one user (actor). Use cases are supported by the computer application by performing one or more services. The functional size of a use case is measured. This size is to be understood as an estimation of the functional of the glue logic between services which are an implementation of business rules.

A business object is a group of data which describes 'things' relevant to the business (person, thing, event, screen, contract etc.). Business objects and their mutual associations are structured in a conceptual business object model. The interaction between business processes and business objects is recognized as service/class relations.

System Domain Elements (BFC's)

Services are the equivalent of the FPA-transactions, and as such have to comply to the definition of the *elementary process.* Services are the implementation of a transaction (i.e. a computer supported activity within a use case). Besides functionality visible to the user is also functionality invoked by non human actors is provided by services. Services are implemented by one or more operations (responsibilities) provided by classes.

A Class is a combination of attributes, their respective values and operations. These elements determine the behavior and responsibilities of the class. In our approach operations are considered as a separate recognizable functional element (BFC) which is counted apart from the attributes.

Operations and transformations are the actual providers of processing required for the realization of services. Operations are the smallest recognizable unit of functionality. Operations represent a significant part of the functionality required by the user and subsequently form a significant part of the functional size. Operations are classified by their nature in 'query'- and 'modify'-operations.

A transformation is a (series of) arithmetic operation(s) that changes input into a formal output result. With the definition of transformations we follow 3D Function Points from Whitmire [16]. Within a use case, transformations often appear as 'uses'-services. Transformations are considered apart from operations because this can facilitate a possible refinement of the technique in the future. It probably enlarges the field of application for this approach to other functional domains.

Types of count

Our sizing approach distinguishes three types of estimation with an increasing degree of accuracy. The count types correspond to purpose and scope described in the objectives.

Domain model count

Preferably filled in with FPA because the vast number of benchmark material has great value. This requires an investigation to the relation between COP's and FP's.

Analysis count

The objective of the analysis count is to measure the amount of functionality required by the user in terms of functional units (COP's), based on the results of the analysis stage.

Counting elements (BFC's)

use case services (computer supported activities)

- significant classes and object structures (shown in the class diagram)
- relation service/class
- transformations

The analysis count results in a list of use cases, with every use case consisting of one or more services with their respective size expressed in COP's. Apart from the use cases the analysis count results in a list of significant classes with their respective functional size, determined from service/class relations.

The counting results are aggregated at the level use cases and classes. Optionally the size of classes participating in an object structure can be aggregated at the level of highest super-class. Through aggregation the counted elements can be mapped into a structure of patterns or sub-components to be realized as development increments. The idea behind the aggregation of sizing results is that for each increment a design time-box estimate can be obtained from the relative size of the sub-component(s) together with a productivity standard for the appropriate development environment.

Design count

The objective of the design count is to provide an accurate measurement of the amount of functionality required by user expressed in COP's, based on the results of the first (logical) design iteration.

Counting elements(BFC's)

- use case services
- classes (directly referenced)
- specification service/class relation (operation)
- transformations

The design count results in an accurate product-size reported in service- and class-functionality. Transformations and their size are reported separately. Although not elaborated in this first proposal it is the intention to develop a tool set with reporting facilities by which (sub-)components or

development increments can be reported as a recognizable unit. Through this facility these units can be matched by class engineers with class- or component-libraries in order to carry out a gross/nett comparison and to determine the nett project size.

Required specifications

For each count type a set of specifications are required which describe the counting elements. The minimal specifications required for each type of count are listed hereafter.

The analysis count requires the following specifications:

- business processes (workflow diagrams, activities/use cases, tasks/services)
- business object model
- class diagram (conceptual view, showing significant classes and object-structures)
- relation service/class (use case model, activity-diagram, collaboration/sequence- diagram)
- [object life cycle diagrams (recommended)]
- [application boundary, interfaces with other applications]

For a design count the same specifications are required as for an analysis count.

- use case models (activity diagram)
- class diagrams and descriptions (describing responsibilities, i.e. attributes, operations)
- collaboration/sequence diagram (interactions between objects)
- [object life cycle diagrams (not necessary, recommended)]
- [application boundary, interfaces with other applications]

Counting structure

The total size of an application is derived by adding the service functionality to the class-functionality. The structure of the application size is shown in figure 4.

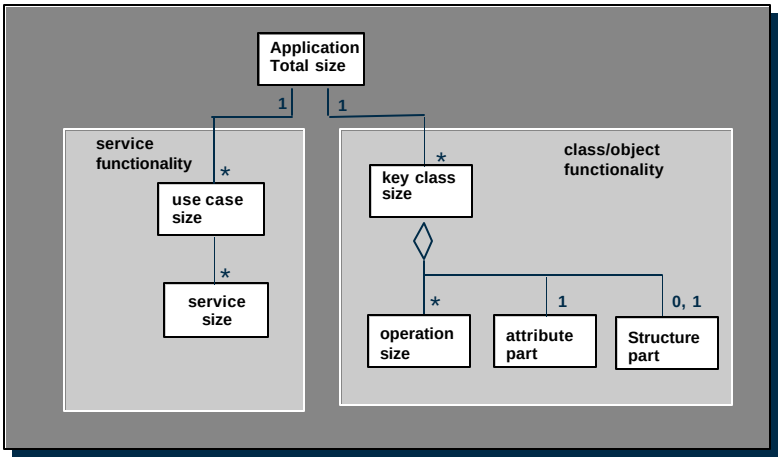


fig. 4 size structure of an OO-application

Valuation of counting elements

One of the preconditions for the development of this estimation technique is to aim for the same proportions as FPA. The reason is to be able to compare results from our size estimations with results of domain model counts in FP's. This implies that there was no attempt to apply mathematical rules or models for the expression of functional size in the unit of measure COP. Instead an extensive

experimental period have been worked through, to derive "weighted values" as a result of interaction with developers on a basis of trial and error. We aimed for a "good feeling" about the perceived amount of functionality and the development effort. Besides "good feeling" the values in the valuation matrices were tuned to the values of the Programming Language Tables of Jones [19] and those collected by Putnam [20]. The Function Point Counting Practices: Case Study 3 [18] has been used for comparison.

Pilots and field experience

Three projects were selected for tryout. Two are to be considered business information systems, one as a combination of control software, embedded software, and recording/ presentation software. Project-1 was a small to medium "tool rental application" for industrial equipment. Project-2 was a "Billing & Customer Care application" for telecom services, the size is qualified as big. Project-3 from which only certain parts were taken for tryout was a complex system for "medical equipment", the size of the entire system is qualified as very big.

The pilots suffered from an undesirable lack of standardization in the functional OO-specifications between different organizations (the UML did not come one day too soon and to our opinion supports the early development stage of "domain modeling" insufficiently). For all three projects an analysis count could be carried out. But only Project-3 was specified in a sufficient degree of precision for a design count. Nevertheless we find the results of the two other projects encouraging because an analysis count gave interesting insight in the measured amount of functionality and the division thereof over the key classes. The overall conclusion of the pilots is that project estimation benefits the most from the analysis count because it requires a limited effort and gives valuable insight about the way the required functionality is spread over the key-classes, without a necessity of too precise specifications as input. The result is a fairly accurate estimation expressed in COP's on which a time-budget for (a part of) the project can be made.

Future research

As mentioned in the abstract the development of this estimation approach has been continued with a focus on component based development. The conceptual model remains the same, but classes are replaced by "components". The next steps have already been taken and a functional sizing approach for components and assemblies, based on the described method is in the pilot stage.

Preliminary counting rules

Process steps for analysis and design counts

The counting process for the analysis and design type counts consists of the following steps.

1. Determine count type.
2. Determine the counting boundary and granularity.
3. Review the specifications of the counting elements with respect to the count type.
4. Identify and valuate use case services.
5. Identify and valuate use case service / class relations.
6. Identify and valuate classes and structures within the domain model.
7. Identify and valuate operations and transformations (design count only)
8. Determine reusable elements (design count only)
9. Determine the total size of the application

Counting rules for analysis and design count

Step 1 Determine count type

This step is a formal one and is based on the purpose of the count. The point in time as well as the availability of the required specifications determine which count type can be carried out.

Step 2 Identify the counting boundary

For this step the rules from Fetcke [1] are adopted. "The view of the OOSE use case model corresponds to the boundary concept of Function Points, as the actors are outside the application and the use cases define the application's functionality." Jacobson [14] calls this boundary the *system delimitation*. The same goes for the comparison between FPA-users and OOSE actors from Fetcke [1]. "Each user of the application has to appear as an actor. Similarly, every other application which communicates with the application under consideration must apply as an actor too." Non-human actors which are part of the counted system are not recognized as a valid actor.

Note that if the underlying system is an assembly of autonomous components a counting boundary may be drawn around each component. All other components except the subject of counting are valid actors in such a situation.

Step 3 Review the specifications of the counting elements

The specifications to be used as input for the count have to be reviewed against a set of minimum requirements. It is required that use case / activity descriptions and business object-/class-diagrams have the same degree of detail and are consistent i.e. contain the same object names. Inconsistencies have to be reported and a decision have to be made whether a count nevertheless can be carried out or not.

Step 4 Identify and valuate use case services

A use case is decomposed in activities which are to be supported by the application through *services*. Services are compared with the definition of an elementary process. All types of services are counted whether they are visible to the user or automated. The amount of functionality a service represents is determined by the number of operations/transformations invoked. It represents the logic as an implementation of business rules and control.

During the use case analysis a list of classes referenced in the use case descriptions have to be build. The class list will be used to determine significant classes during class-diagram analysis.

Counting rules (Analysis count)

1. Decompose the use case activities into services. Services have to comply to the definition of an elementary process (i.e. must leave the system in a consistent state after execution).
2. Count 2 points per service; summarize per use case.

Counting rules (Design count)

1. Decompose the use case activities into services. Services have to comply to the definition of an elementary process.
2. Valuate services using the Service matrix; summarize per use case.

Services valuation matrix

#Operations/ transformations	1	2 - 3	>3
COP's	1	2	4

Step 5 Identify and valuate service / class relations (analysis count only)

In this step the relation between services and classes are investigated. At the level of consideration of an analysis count, only the existence of such a relation is relevant. The investigation of this relation is based on CRC-card modeling [13, 15] which we consider as a precursor of collaboration described in UML [21]. This approach does match remarkably well with the level of consideration during an analysis count. The high level classes are being defined at this stage and so are the use cases and containing services. During the analysis count a cross reference between services and responsible classes is drawn up. During a design count the relation between services and classes are identified as operations and qualified by their nature.

Note: in an environment with typed classes, only entity type classes are considered. An other way to distinguish classes to be considered and classes which are not is a differentiation in key-classes and supporting classes as indicated by Lorenz & Kidd [4]. "A key-class is one that would cause great difficulty in developing and maintaining the system if it did not exist". Key-classes are the carriers to which functionality is assigned.

Counting rules service/class relations (Analysis count)

1. Analyze the services within a use case and relate the service to all classes that collaborate to provide (parts of) the service.
2. Count 3 points as responsibility-functionality for every unique relation service/class and accumulate to the appropriate class.

Counting rules Transformations (Analysis count)

1. Analyze use case activities to discover transformations (in the context of a service), and count 5 COP's for each transformation.
2. The amount of functionality resulting from transformations is accumulated separately from service-functionality.

Step 6 Identification of business objects, classes, object structures

The Business Object Model is used as a reference model for the interpretation of a more detailed Class Diagram. The following counting elements determine the functional size: significant classes, structures of classes (object-structures). As a starting point class diagrams are determined after a possible system-wide modeling has been carried out, in order to discover significant collaborating classes. The significance of classes is in proportion with the level of detail of the modeling specifications.

Significance implies:

- must be modeled in the conceptual class-diagram
- must be referred in one or more use cases (directly or as a collaborating class)
- can be considered a key-class

All classes are counted for the attribute aspect and separately for the structure aspect if applicable. The counting rules are explained hereafter. Apart from elementary sub-classes in a aggregation fairly all (entity type) classes turn out to be significant (and key-class).

Classes

A class is recognized as super- or sub-class depending on the position in a structure. All classes shown in the class-diagram are counted for their attribute-part regardless of their position in a structure. The classes are counted for their own number of attributes. Inherited attributes are not regarded. The structure aspect of a class is counted depending on the type of structure they are part of.

Counting rules for the attribute aspect (both types of count)

The attribute-aspect is counted by the number of attributes according to the matrix mentioned hereafter. If the number of attributes is not specified 5 COP's are counted.

Class attribute valuation matrix

Attribute part	# Attributes	<3	3 - 6	>6
	COP's	2	5	7

Object structures

Object structures implying functionality to the user are counted. Generalization/ specialization and aggregation/composition are such structures. Cardinality associations are ignored as is the case in FPA.

Class-diagrams are determined according to the following table. Every significant class have to be processed according to the table rules.

Table 1

Q1	Is the class a generalization?	Y	go to A1 in table 2
		N	-
Q2	Is de class an elementary specialization?	Y	Apply counting rule for gen./spec. structure
		N	-
Q3	Is the class an aggregation/composition?	Y	Apply counting rule for aggr./comp. structure
		N	-

Table 2

A1	Has the structure from which the actual class is part of already been counted?	Y	no action
		N	- apply counting rule gen./spec. - return to Q3 in table 1

Remark: it should be noticed that classes can be both an aggregation structure and a generalization structure. Both aspects represent functionality and are taken into account.

Counting rules for Generalization/specialization structures (both count types)

1. The entire structure from which the significant class is part of is recognized as a *significant structure*. Count all structure levels up to and including the highest super-class. The structure obtains the name of the highest super-class.
2. Valuation: 3 COP's per structure level

Counting rules Aggregation/composition structures (both count types)

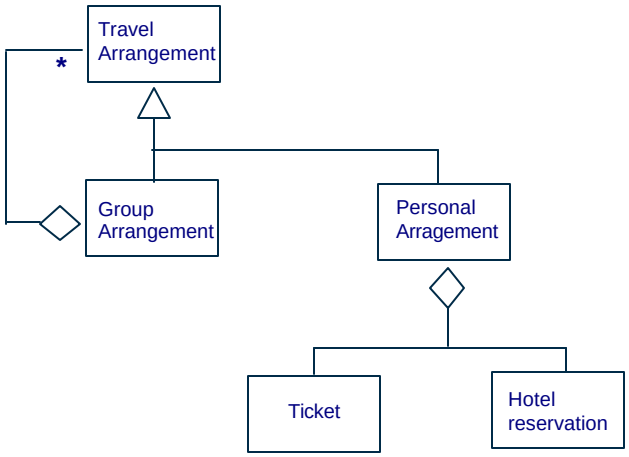
1. Count all the component sub-classes which are part of the structure.
2. Valuation: 2 COP's per sub-class

Object-structure valuation matrix

Structure part	Association type	Gen./Spec. Aggr./Comp.
	COP's	#Lvls*3 #Sub-cl*2

Total class valuation = $COP's_{structure\text{-}part} + COP's_{attribute\text{-}part}$

Example



Provided all classes shown in the example are significant. The class diagram shows three object structures, which are counted as follows:

Structure name	*type*	*# levels*	*# sub-classes*	*COP's*
Travel Arrangement	Gen./Spec.	1	-	3
Group Arrangement	Aggr./Comp.	-	1	2
Personal Arrangement	Aggr./Comp.	-	2	4

[Classes which represent two different structures are counted for twice (for each structure). This does not imply a double count because different elements are counted (#levels as opposed to #sub-classes)]

[*Constraints and invariants*
Constraints do have a significant effect on complexity and as such represent functionality. It is assumed that the number of constraints is reasonably in line with the number of attributes, and as such are represented by the number of attributes. This approach is in the line of thought of FPA. The 'attribute'-part of the class therefore represents the functionality from both the attributes and their constraints.]

Step 7 Identify and valuate operations & transformations (design count only)

The relation between services and classes are differentiated by identifying the nature of the requested operation. The nature of the operation is defined using two criteria: type of operation (Query/Modifying) and whether or not collaborating classes are involved.

Implicit operations

Some operations can be assumed to be mandatory for each class and may not be formally identified. These are called *implicit operations*. Well-known important implicit operations are *create*, *destroy*, *update* and *read*. Although these operations may be automatically generated by the development environment, they have to be formally identified, because they represent significant functionality to the user. The standard implicit operations are classified as reusable elements, which implies no effort. Implicit operations with pre- and post-conditions are not trivial and have to be counted anyway because the business rules represent FUR's.

Transformations

Transformations are valuated using only one criteria: the number of collaborating classes. The complexity aspects from arithmetic expressions are expressed by applying a higher number op COP's than for operations.

Counting rules operations

1. For all services identify the relation between the service and the first responsible class as one or more operations.
2. The value of an operation is determined by means of the valuation matrix as follows:

- the type of operation determines the row (if the operation changes an object it is recognized as a Modifying operation, else it is recognized as a Query operation.
- collaborating class(es) determine the appropriate column.

3. Add the number of points to the appropriate class.

Counting rules transformations

1. Analyze use case activities to discover transformations (in the context of a service).
2. Valuate transformations using the Operation/transformation matrix.

3. Accumulate the amount of functionality resulting from transformations separately from service-functionality.

Operation/transformation valuation matrix

Collaborating Classes	N	Y
Query	2	3
Modify	3	4
Transformation	4	6

Step 8 Determine reusable elements

This step is still to be developed and is strongly dependent on the development environment. A case-tool furnished with a class-catalogue or repository has to be considered a prerequisite. The results of a design count will be drawn up in a way that groups of functionality can be considered separately. A computerized tool is recommended but not available at this moment.

Step 9 Determine the total size of the application

The total size of the application is derived by adding the service functionality to the class-functionality.

Acknowledgments

I wish to thank the project teams at IQUIP Informatica B.V., Netherlands for their cooperation and explanation of their development process. Special thanks is granted to René van Oosterwijk of Informatie Beheer Groep, Groningen, Netherlands for his contributions and sparring sessions.

References

[1] Thomas Fetcke et al., "Mapping the OO-Jacobson Approach into Function Point Analysis",*Proceedings of Tools-23 '97* Santa Barbara, CA

[2] S.R. Chidamber and C.F. Kemerer, "Towards a Metrics Suite for Object-Oriented Design", *OOPSLA '91 Conference Proceedings, Special Issue of SIGPLAN Notices*, Vol. 26 No. 11, November 1991, pp. 197 - 211.

[3] S.R. Chidamber and C.F. Kemerer, *Managerial Use of Metrics for Object Oriented Software: an exploratory analysis*, KGSB Working paper no. 750, Pittsburgh, PA 15260, 1997.

[4] Mark Lorenz, Jeff Kidd , *Object-Oriented Software Metrics* Prentice Hall Object-Oriented Series, 1994

[5] Philip Haynes, Tim Menzies, Geoffrey Phipps, *Using the Size of Classes and Methods as the Basis for Early Effort Prediction; Empirical Observations, Initial Application; A Practitioners Experience Report*, September 1995

[6] Sita Ramakrishnan, *An Ongoing Experiment in O-O Software Process and Product Measurements*, TR95/22, Dept. Software Development, Monash University, Australia.

[7] Tim Hastings, *Adapting Function Points to contemporary software systems: a review of proposals*, P95-5, Dept. Software Development, Monash University, Australia.

[8] L. Laranjeira, Software Size Estimation of Object-Oriented Systems, *IEEE Transactions on Software Engineering*, Vol. 167, No. 5, May 1990, pp. 510 - 522.

[9] Tony Stockman, Hua Zhao, "Software Sizing for OO Software Development- Object Function Point Analysis", *Proceedings GSE Conference*, Berlin, March 1996.

[10] B. Catherwood, M. Sood, F. Armour, "Continued Experiences Measuring Object Oriented System Size", Proceedings ESCOM'97, Berlin, May, 1997.

[11] Georges Teologlou, "Measuring object oriented software with predictive object points", *Project Control for Software Quality*, (proceedings of ESCOM SCOPE 99 Conference, May 1999, Herstmonceux, England, Shaker Publishing, 1999

[12] International Organization for Standardization, *Information Technology - Software Measurement - Functional size measurement - Part 1: Definition of concepts*, ISO/IEC 14143-1: 1998.

[13] Martin Fowler, Kendall Scott, UML Distilled, Applying the standard Object Modeling Language, Addison Wesley, 1998

[14] Ivar Jacobson, M. Christerson, at al., *Object-Oriented Software Engineering 'A use case driven approach'*, Addison-Wesley, 1992

[15] Scott W. Ambler, *CRC Modeling: Bridging the Communication Gap between Developers and Users*, A White Paper, 1997

[16] Scott A. Whitmire, *3D Function Points: Applications for Object-Oriented Software* Boeing Commercial Airplane Group, PO Box 3707 MS 6C-FL, Seattle Washington 98124-2207.

[17] International Function Point Users Group, *Counting Practice Manual Rel. 4.0,* Westerville, OH 43081-4899, 1994.

[18] International Function Point Users Group, *Function Point Counting Practices: Case Study 3*, Westerville, OH 43081-4899, 1994.

[19] Capers Jones, Programming Languages Table, Release 8.2, March 1996, Software Productivity Research.

[20] Lawrence H. Putnam, Ware Myers, Measures for Excellence, Reliable Software on Time, within Budget, Yourdon Press, P T R Prentice-Hall, 1992.

[21] Hans-Erik Eriksson, Magnus Penker, *UML Toolkit*, John Wiley & Sons, Inc., 1998.