Functional Size Measurement of Multi-Layer Object-Oriented Conceptual Models

Geert Poels [1, 2]

[1] Department of Management Information, Operations Management, and Technology Policy
Faculty of Economics and Business Administration
Ghent University, Hoveniersberg 24, 9000 Gent, Belgium
geert.poels@ugent.be

[2] Centre for Industrial Management, Katholieke Universiteit Leuven
Celestijnenlaan 300, 3010 Heverlee, Belgium
geert.poels@econ.kuleuven.ac.be

Abstract. This paper builds on previous work showing a way to map the concepts used in object-oriented business domain modelling onto (a subset of) the concepts used by the COSMIC Full Function Points (COSMIC-FFP) functional size measurement method for modelling and sizing a software system from the point of view of its functional user requirements. In this paper we present a refined set of measurement rules resulting from a careful revision of our previous proposal, based on 'field trials' and feedback from function points experts. The main contribution of the paper is, however, an extended set of rules to be used when applying COSMIC-FFP to multi-layer conceptual models, including at least an enterprise layer and, on top of this, an information system services layer.

1 Introduction

This paper presents results of ongoing research into the size measurement and cost estimation of software systems that are developed using the Model-Driven Architecture framework envisioned by the Object Management Group [1]. This approach to software development prescribes the construction of software systems through the transformation, via platform-independent design patterns and platform-dependent implementation schemes, of computation-independent conceptual representations of the enterprise and its (required) information system(s).

First research results, consisting of a set of rules for applying COSMIC-FFP [2], which is a generic functional size measurement method, to object-oriented business domain models were published in [3]. In particular, we showed that the meta-model of methods that take an event-based approach to business domain modelling (e.g. OO-Method [4], MERODE [5]), can naturally be mapped onto (a subset of) the abstract model of functional user requirements that is used by COSMIC-FFP as the basis for measuring the system's functional size.

In this paper we extend this previous work by showing how to measure the functional size of models that are organised in a layered architecture. We present specific COSMIC-FFP rules for the information system services model, which is a layer on top of the enterprise layer (i.e. the business domain model) in a layered conceptual model. The information system services model is used to model information system functionality related to the information needs of system end-users (e.g. management reports, business documents), the support of system end-users involved in business operations (e.g. the input of business transaction data) and the support of system (or business) management activities (e.g. system's (or business) performance monitoring).

We also present a revised and refined version of the COSMIC-FFP rules for business domain models proposed in [3] and show how to separately size the enterprise and information system service models.

Although COSMIC-FFP allows sizing separate layers of a software system, the application of the COSMIC-FFP concept of software layer has, to the best of our knowledge, not been demonstrated yet, neither are we aware of measurement rules or guidelines for specific architectural paradigms. The theoretical contribution of our work is therefore a proof of concept of measuring multi-layer conceptual models by separately mapping the meta-models of interacting conceptual model layers onto (overlapping) subsets of the COSMIC-FFP model.

To substantiate the proposed mapping of concepts and to exemplify the proposed measurement rules we use MERODE [5], a semi-formal conceptual modelling method that prescribes a layered object model of system specifications and a CASE-supported model-driven (i.e. transformation-based) approach to systems development. Hence, the practical contribution of our research is a MERODE-specific set of rules to apply COSMIC-FFP to multi-layer conceptual models. Once such a set of rules has been established, the sizing of MERODE conceptual models can be automated in order to maximise the efficiency of the measurement process and to assure the quality (e.g. reliability, consistency) of the measurement results.

After a brief review of related work in section 2, we list in section 3 the main concepts of COSMIC-FFP, before proceeding to a discussion of layered conceptual models in section 4. In section 5 we map the modelling concepts of the enterprise and information system service models onto the concepts of the abstract COSMIC-FFP model and derive specific functional size measurement rules from this mapping. Conclusions are presented in section 6.

2 Related Work

The mapping of object-oriented modelling concepts onto the abstract COSMIC-FFP model of functional user requirements, resulting in the proposal of more specific measurement rules than the general rules provided by the COSMIC-FFP measurement manual [2], has received some research attention lately.

Bévo et al. [6] have mapped the concepts used in UML class diagrams and use case diagrams onto the abstract COSMIC-FFP model, in order to measure the functional size of software based on high-level specifications. Similar work has been done by Jenner for UML sequence diagrams [7]. Both proposals focus on finding an appropriate mapping for UML modelling concepts, without reference to the modelling method that is used. An equivalence to the software layer concept in COSMIC-FFP is only suggested by Jenner, in the form of ‘swimlanes’ in sequence diagrams. The author does, however, not present an example of a sequence diagram with ‘swimlanes’, and in a later publication on the automation of COSMIC-FFP measurement of sequence diagrams [8], this idea is not elaborated any further.

Not unlike Jenner, the work of Diab et al. [9], [10] aims at developing a set of automatable rules for applying COSMIC-FFP to object-oriented specifications. It is argued that such automation support will significantly reduce the measurement variance and cost that is usually observed when applying ‘function points’-based measurement methods. The mapping and rules proposed by Diab et al. are specific to the Real-time Object-Oriented Modelling (ROOM) method, as supported by the Rational Rose RealTime (RRRT) toolset, thereby distinguishing this work from the aforementioned work on the modelling language UML. In their first proposal [9], the authors use a type of statechart diagram, called ROOMcharts, as the primary basis for functional size measurement. In their follow-up work [10] a complete RRRT model is required. In this latter work, the concept of a COSMIC-FFP software layer is interpreted as a set of capsules (i.e. active objects) in a RRRT model. It is, however, not mentioned how such sets should be delineated in a model.

The application of COSMIC-FFP functional size measurement to multi-layer conceptual models, resulting from an architectural decomposition guided by well-defined layering principles, is the most distinguishing feature of our work compared to the works presented in this section.

3 The COSMIC-FFP Model

The main COSMIC-FFP concepts, relevant for this paper, are listed below. Where necessary, concepts will be further clarified in the rest of the paper.

- A *scope of measurement* is defined to delimit the software system to be sized and to separate this system from its environment.
- Within this scope of measurement, the software system can be broken down into *pieces of software*, either as separate *software layers* or as *peer items* within a software layer.
- The collection of functional user requirements for a piece of software allows identifying one or more *users* of that piece of software.
- These users are in the environment of the piece of software and interact with it across a *boundary*.
- The collection of functional user requirements for a piece of software is decomposed into one or more *functional processes*.
- On the piece of software side of the boundary there is *persistent storage*, i.e. continuously accessible storage that enables functional processes to store/retrieve data beyond their individual lives.
- Any functional process can be decomposed into two or more *data movements*.
- A data movement is a sub-process of a functional process that moves a *data group* (one or more *data attributes*) about a single *object of interest*, and which may include some associated data manipulation operations.

- There are four *types of data movement*:
  - An *entry* moves a data group from a user across the boundary into the piece of software.
  - An *exit* moves a data group from the piece of software across the boundary to a user.
  - A *write* sends a data group from the piece of software to persistent storage.
  - A *read* retrieves a data group from persistent storage for the piece of software.
- The *COSMIC-FFP measurement standard*, 1 COSMIC Functional Size Unit (CFSU), is defined as being equal to a single data movement.
- The *functional size* of a piece of software is the sum of the functional sizes of its constituent functional processes; the functional size of a functional process is the sum of the functional sizes of its constituent data movements; the functional size of a data movement is, by definition, 1 CFSU.

4 Multi-Layer Object-Oriented Conceptual Models

Although the conceptual model architecture described in this section is specific to MERODE, its underlying layering principles are sufficiently general to be observed/applied in other object-oriented modelling methods.

We first discuss the general model architecture as in [5] and next present a system's view on the enterprise model and the information system services model, which are the layers for which COSMIC-FFP measurement rules are proposed in this paper.

4.1 A Layered Architecture for Conceptual Models

A multi-layer conceptual model is obtained through the partitioning of the object model that is a conceptual representation of the enterprise and its (required) information system. The basic principle guiding this partitioning is the expected change rate of the things that are represented in the conceptual model.

One partition is formed by the business domain model (also called enterprise model), which represents business entities, business events, their interactions, and the rules governing these interactions (i.e. business rules). This is the most stable part of the conceptual model as it describes 'real world' phenomena that are also observed in the absence of an information system. The modelling diagrams, formalisms and techniques used in the process of constructing this enterprise model include class diagrams (using UML notation), an object-event table showing which (types of) business events affect which (types of) business domain objects, and state transition diagrams to model object behaviour.

Another conceptual model partition consists of the representation of the required information system services. The information system services model specifies the end-user facilities to generate events from business transactions or activities and transmit these to the enterprise model. It also represents the facilities to satisfy the end-user information needs. This part of the conceptual model is less stable than the enterprise model as the functions that an information system must fulfil depend heavily on the particular work organisation in the enterprise and the specific end-user information needs.

The conceptual model can be further extended with business process models and workflow models. Information system end-users involved in workflow activities invoke business events and satisfy their information needs through the user interface model. A third partition therefore represents the required facilities to trigger, interrupt and resume the execution of the information system services. This user interface model also captures presentation aspects, which are volatile as they depend on end-user preferences (which are dynamic by nature).

In object-oriented approaches to conceptual modelling, like MERODE, the basic modelling concept is the object (type or class). The different partitions of the conceptual model can be organised as a hierarchy of layers of objects such that the enterprise objects are in the lowest layer, the information system service objects are in the middle layer, and the user interface objects are in the highest layer. Objects are only allowed to invoke the functionality of objects in the same layer or in a lower layer. They are unaware of objects in higher layers. That way, objects are prevented to depend upon less stable objects (found in higher layers), ensuring a strict control on the propagation of changes.

To model the interaction between service objects and enterprise objects MERODE uses two object communication mechanisms:

- To transmit business events to the enterprise model use is made of the event broadcasting mechanism meaning that, from a conceptual modelling point of view, service objects 'broadcast' event messages without knowing where exactly these messages will be received in the enterprise model. At implementation time, an event handling layer can be introduced to act as 'middleware' between the services and enterprise layers in the information system. The objects in this intermediate layer are responsible for invoking the right class methods, checking method preconditions, etc.
- The state vector inspection mechanism allows service objects to access an enterprise object's attributes (whose values form, collectively, the enterprise object's state vector). The realisation of this inspection mechanism in the object system (for instance via accessor or selector methods in the class definitions of enterprise objects) is again an implementation issue, which should not be determined during the conceptual modelling process.

4.2 A Taxonomy of Service Objects

Fig. 1 depicts the information system services and enterprise layers of the conceptual model as a cybernetic dynamic system (as defined in Systems Theory). The enterprise objects form the processing component of the system. These persistent objects are responsible for processing business events and maintaining (i.e. creating, updating, destroying) business data, using the services offered by a database management system via a database mediator like an object broker (which is outside the scope of the conceptual model).

The information system services model contains three types of non-persistent service objects:

- *Input objects* collect data on business transactions and activities, and generate one or more business events, which are transmitted to the enterprise model. They are allowed to inspect the state vector of enterprise objects. Their functionality is invoked by a timer or by end-users via user interface objects when performing business (or workflow) tasks. These elements are in the environment of the system as they belong to conceptual model layers that are not considered here.
- *Output objects* extract information from business data that are obtained through state vector inspections of enterprise objects. The required information products are sent to end-users via user interface objects, upon end-user request or, automatically, upon occurrence of a business (or clock) event.
- *Control objects* are similar to input objects, except that their functionality is invoked by managerial end-users in order to control the performance of the system (or the business).

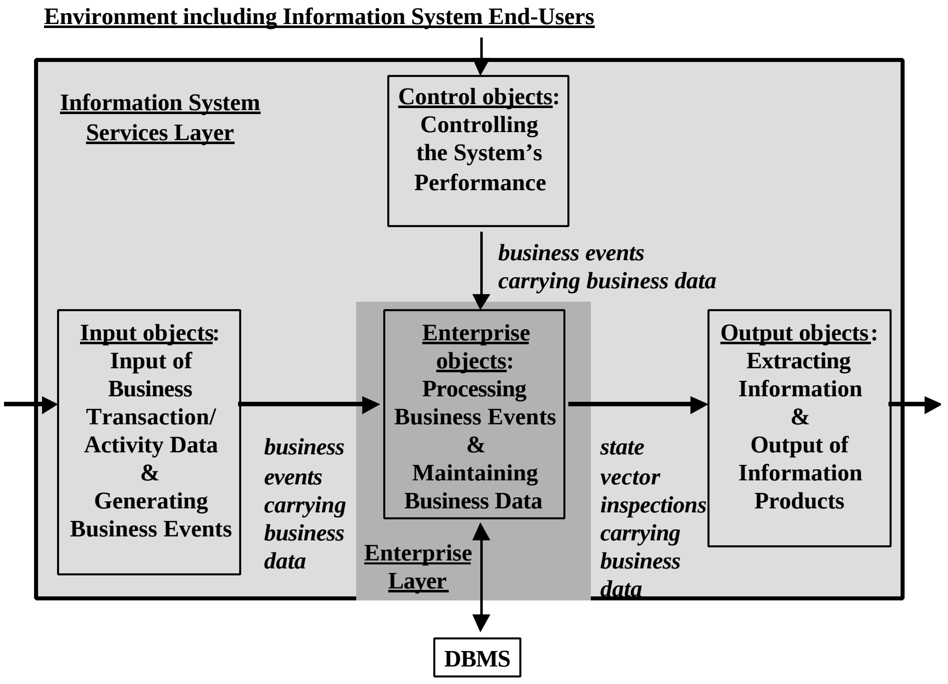
Environment including Information System End-Users
Information System Services Layer
Control objects: Controlling the System's Performance
business events carrying business data
Input objects: Input of Business Transaction/ Activity Data & Generating Business Events
business events carrying business data
Enterprise objects: Processing Business Events & Maintaining Business Data
Enterprise Layer
state vector inspections carrying business data
Output objects: Extracting Information & Output of Information Products
DBMS

Fig. 1. System's view on enterprise model and IS services model [11]

In practice, services required from an information system often involve *interactive functions*, which combine input and output facilities (e.g. input of modified customer data, after having selected a customer by means of a pick-list). To keep our taxonomy simple, we assume that such functions are modelled by means of combinations of input and output objects, and that the required dialogue and sequencing aspects are modelled in higher layers of the conceptual model.

Tables 1, 2 and 3 provide further details on input, output, and control objects by listing their respective triggers and functions.[1]

Table 1. Properties of input objects

Triggers	(1) Input service request message from UI object (2) Input service request message from clock object
Functions	(1) Broadcast business event message (2) Send error/confirmation message to UI object[2]

Table 2. Properties of output objects

Triggers	(1) Output service request message from UI object (2) Output service request message from clock object (3) Business event message from input/control object
Functions	(1) Send required information product to UI object (2) Broadcast IS event message[3] (3) Send error/confirmation message to UI object

Table 3. Properties of control objects

Triggers	(1) Control service request message from UI object
Functions	(1) Broadcast business event message (2) Send error/confirmation message to UI object

5 Measuring Conceptual Model Layers

We first review the COSMIC-FFP rules for the enterprise model as proposed in [3]. Since their proposal, these rules have been refined based on experiences gained by applying them to a benchmark functional size measurement case-study [12].

Next, new COSMIC-FFP rules are presented for the information system services layer in a conceptual model architecture.

[1] It should be noted that the taxonomy presented here does not cover the entire range of information system services supported by MERODE (see [5] for a more detailed discussion). We believe, however, that the main concepts that can be generalized to the class of model-driven conceptual modeling methods, are included in our framework.

[2] A return message might be sent to the requesting UI object in order to confirm the delivery of the service or to report errors.

[3] Information system events are events that are broadcasted to a subset of the persistent objects in the enterprise model (called *information objects*) to guarantee the correct functioning of the output facilities. An example is the request to print a statement for a checking-account [5]. Such a statement lists all deposit and withdrawal events since the last time this output facility was invoked. Hence, printing the statement is an information system event carrying data (e.g. a timestamp) that must be stored by persistent objects (i.e. information objects).

5.1 COSMIC-FFP Rules for the Enterprise Model

Table 4 establishes an equivalence between the concepts of the COSMIC-FFP model and the enterprise model, when used as a separate layer within an object-oriented conceptual model architecture.

Table 4. COSMIC-FFP and business domain modelling

COSMIC-FFP	***MERODE***
Scope of measurement	Enterprise layer of the conceptual model architecture
Users	Objects of the information system services layer
Persistent storage	Storage devices accessible via database mediator and DBMS
Functional process	Set of class methods, over all enterprise objects, that are invoked by the occurrence of a type of business event
Entry	Class method invocation by a business event message
Exit	-
Read	Retrieval of enterprise object state vector
Write	Storage of enterprise object state vector

Some further clarifications and CFSU counting rules (in *italic*) are presented below:

– If the scope of measurement is the entire conceptual model, then the enterprise model qualifies as a software layer. According to the COSMIC-FFP measurement manual [2], one software layer can be the user of another software layer. Here we consider the objects of the information system services model as users of the functionality offered by the enterprise model.
– According to the measurement manual, a functional process is triggered by an event (type) and is complete when it has executed all that is required to be done in response to the triggering event (type). Hence, for each business event type a functional process is identified that includes all class methods that can be invoked by occurrences of this type of event.
– The objects of interest about which data is moved in the functional processes are business events and enterprise objects.
– Business event message arguments carry data about business events from input objects to enterprise objects. These message arguments correspond to the parameters in the signature of the class methods that are invoked by the event. Hence, *an entry data movement is identified for each method in the enterprise object classes that can be invoked by a business event*.
– Although, conceptually, state vector inspections move data from the piece of software under consideration (i.e. the enterprise model) to the users (i.e. service objects), their corresponding messages are only specified in the information system services model and their implementation (e.g. through accessor or selector methods) is not a conceptual modelling issue. Therefore, *no exit data movements are identified*.
– The retrieval of an enterprise object's state vector from persistent storage qualifies as a read data movement. Such a read data movement occurs whenever the method body or preconditions need to know the value of at least one enterprise

object attribute. *The number of read data movements for a class method is equal to the number of different enterprise object state vectors that must be accessed*.

– An enterprise object's state vector must be stored whenever the value of at least one of its attributes is updated in a method body. Hence, *if there is at least one update operation in the method body, a write data movement is identified*.[4]

5.2 COSMIC-FFP Rules for the Information System Services Model

Table 5 is similar to table 4, but now shows the mapping of COSMIC-FFP concepts into the MERODE information system services model.

Table 5. COSMIC-FFP and information systems modelling

COSMIC-FFP	***MERODE***
Scope of measurement	Information system services layer of the conceptual model architecture
Users	Objects of the user interface layer
Persistent storage	Persistent objects in the enterprise layer
Functional process	A non-persistent service object invoked by an input, output or control service request message or (for output objects only) a business event occurrence
Entry	Input/Control objects: Service method invocation by a service request message Output objects: Service method invocation by a service request message or a business event message
Exit	Input/Control objects: Confirmation/Error message Output objects: - Confirmation/Error message - Information product (return) message
Read	State vector inspection of persistent object in enterprise layer
Write	Input/Control objects: Broadcasting of a business event message Output objects: Broadcasting of an information system event message

Further clarifications of the mapping and CFSU counting rules (in *italic*) are presented below:

– Again, if the scope of measurement is the entire conceptual model, then the information system services model qualifies as a software layer. The user

[4] Whereas the methods in an enterprise object class definition are allowed to access the attributes of enterprise objects of different types (following the links in the class diagram), they may only modify the attribute values of their own enterprise objects (i.e. of the type described in the class definition).

interface objects can be considered as users of the service objects, which are in turn users of the enterprise objects. The use of the COSMIC-FFP layer concept allows identifying entry/exit data movements across the boundary between the user (i.e. the user interface objects) and the piece of software considered (i.e. the information system services model).

- The collection of input/control objects and the collection of output objects can be considered as peer items within the information system services layer. This point of view allows identifying the transmission of business event messages from input/control objects to output objects as data movements.
- As service objects are not persistent, they cannot store business or information system data beyond their lives. Therefore, conceptually, they use the storage facilities offered by the persistent objects in the enterprise layer. This perspective allows identifying read/write data movements from/to the enterprise model.
- Due to their non-persistent nature, service objects offer their input/output/control functionality by means of, usually, a single method that is invoked by a service request message or (for some output objects) a business event message.[5] For simplicity's sake we do not further distinguish between service object and method. They are the functional processes within the information system services model.
- The objects of interest about which data is moved in the functional processes include business transactions, business events, enterprise objects, information requests, information products, and information system events.
- The following data movements occur within the functional processes of the input/control objects:
  - Input/control service request messages carry data on business transactions or (managerial) end-user information system tasks. Hence, *an entry data movement is identified for each input or control object*.
  - Confirmation/error messages may be sent in response to service requests. We follow the COSMIC-FFP convention by *counting one exit data movement for each input or control object that can send confirmation/error messages* (regardless of the number of different messages that can possible be sent) [2].
  - *Within each input or control object, every state vector inspection of a persistent object in the enterprise layer counts as a read data movement*.
  - *Each business event message that is generated by an input or control object counts as a write data movement*. Note that because of the event broadcasting mechanism, each message is counted only once, regardless of the number of objects that receive the message.
- The data movements that occur within the functional processes of the output functions are the following:
  - *There is an entry data movement for each output object*, either as a consequence of an output service request message or a business event message. In either case data is moved from another piece of software (layer or peer item) into the output functional process.
  - Again, *we count one exit data movement for each output object that can send confirmation/error messages*. Moreover, *there is one exit data movement for*

[5] At implementation time, classes can be defined with facilities to create and destroy service objects. The main facility in such a class definition would be the service method that realizes the input/output/control functionality offered by the service object.

each output object, representing the transmission of the required information to the user interface.

– *Within each output object, every state vector inspection of a persistent object in the enterprise layer counts as a read data movement.*
– *Each information system event message that is generated by an output object counts as a write data movement.*

A Brief Note on Additivity of Functional Size Values. The COSMIC-FFP measurement manual [2] advises not to sum the functional size values of different software layers. This is a pragmatic guideline following the observation that different layers might be implemented using different technologies, and consequently size-based effort estimation is best performed at the level of the software layer. It should be noted further that there is no danger of 'double counting' functionality as long as each layer has been properly sized. That is exactly the reason why there are no exit data movements in the enterprise layer. In the conceptual model of the business domain no specific functionality to handle state vector inspections (e.g. by means of accessor methods) should be defined.

6 Conclusions

In this paper we showed the application of COSMIC-FFP to layered conceptual models. More in particular, we proposed a mapping of concepts and a set of counting rules to apply COSMIC-FFP functional size measurement to the enterprise and information system services layers in a multi-layer object-oriented conceptual model.

We realise that the research results obtained so far are preliminary and only based on argumentation. Therefore a series of validation studies based on an evaluation model for functional size methods [13], [14] and including demonstration proofs, formal proofs [15], rule verification by function points experts, and controlled experimentation has been planned. The evaluation of our proposal following this approach is ongoing research as well as a topic for continued work.

Our future work further involves extending the rule set to other layers in the conceptual modelling architecture.

References

[1] OMG: Model Driven Architecture. OMG 01-07-01, Object Management Group (2001)

[2] Abran, A., Desharnais, J.-M., Oligny, S., St-Pierre, D., Symons, C.: COSMIC-FFP Measurement Manual, Version 2.1. The Common Software Measurement International Consortium (2001)

[3] Poels, G.: A Functional Size Measurement Method for Event-Based Object-Oriented Enterprise Models. In: Piattini, M., Filipe, J., Braz, J. (eds): Enterprise Information Systems. Vol. IV. Kluwer Academic Publishers, Dordrecht (2002) 210-218

[4] Pastor, O., Gomez, J., Insfran, E., Pelechano, V.: The OO-Method approach for information systems modelling: from object-oriented conceptual modelling to automated programming. Information Systems 26 (2001) 507-534

[5] Snoeck, M., Dedene, G., Verhelst, M., Depuydt, A.-M.: Object-Oriented Enterprise Modelling with MERODE. Leuven University Press, Leuven (1999)

[6] Bévo, V., Lévesque, G., Abran, A.: Application de la méthode FFP à partir d'une spécification selon la notation UML: compte rendu des premiers essais d'application et questions. In: Proc. 9th Int. Workshop Software Measurement. Lac Supérieur, Canada (1999) 230-242

[7] Jenner, M.S.: COSMIC-FFP 2.0 and UML: Estimation of the Size of a System Specified in UML - Problems of Granularity. In: Proc. 4th Eur. Conf. Software Measurement and ICT Control. Heidelberg (2001) 173-184

[8] Jenner, M.S.: Automation of Counting of Functional Size Using COSMIC-FFP in UML. In: Proc. 12th Int. Workshop Software Measurement. Magdeburg (2002) 43-51

[9] Diab, H., Frappier, M., St-Denis, R.: A Formal Definition of COSMIC-FFP for Automated Measurement of ROOM Specifications. In: Proc. 4th Eur. Conf. Software Measurement and ICT Control. Heidelberg (2001) 185-196

[10] Diab, H., Koukane, F., Frappier, M., St-Denis, R.: McRose: Functional Size Measurement of Rational Rose RealTime. In: Proc. 6th Int. ECOOP Workshop Quantitative Approaches in OO Software Eng. Malaga (2002) 15-24

[11] Poels, G.: Functional Size Measurement of Layered Conceptual Models. In: Proc. 5th Int. Conf. Enterprise Information Systems. Angers (2003) 411-416

[12] Fetcke, T.: The Warehouse Software Portfolio: A Case Study in Functional Size Measurement. Report No. 1999-20. Software Engineering Management Research Laboratory, Université du Québec à Montréal (1999)

[13] Moody, D., Abrahao, S., Pastor, O.: Comparative Evaluation of Software Estimation Methods: An Experimental Analysis. In: Proc. 1st Int. Workshop Software Quality and Estimation. Denia, Spain (2002) 49-58

[14] Abrahao, S., Condori, N., Pastor, O.: An Experimental Design for Evaluating Functional Size Methods. In: Proc. 2nd Int. Workshop Software Quality and Estimation. Denia, Spain (2003)

[15] Poels, G.: Definition and Validation of a COSMIC-FFP Functional Size Measure for Object-Oriented Systems. In: Proc. 7th Int. ECOOP Workshop Quantitative Approaches OO Software Eng. Darmstadt (2003)