Quantifying Functional Reuse from Object Oriented Requirements Specifications

Nelly Condori-Fernández[1], Oscar Pastor[1], Maya Daneva[2], Alain Abran[3], Jaelson Castro[4]
[1] *Centro de Investigación en Métodos de Producción de Software*
Universidad Politécnica de Valencia, Valencia-Spain
{nelly,opastor}@dsic.upv.es

[2]*University of Twente*
Drienerlolaan 5, 7522 NB Enschede, The Netherlands
M.Daneva@ewi.utwente.nl

[3]*École de Technologie Supérieure ETS*
1100 Notre-Dame Ouest, Montreal, Canada H3C 1K3
alain.Abran@etsmtl.ca

[3]*Universidade Federal de Pernambuco*
Departamento de Informática, Recife Brasil
jbc@cin.ufpe.br

Abstract

Software reuse is essential in improving efficiency and productivity in the software development process. This paper analyses reuse within requirements engineering phase by taking and adapting a standard functional size measurement method, COSMIC FFP. Our proposal attempts to quantify reusability from Object Oriented requirements specifications by identifying potential primitives with a high level of reusability and applying a reuse indicator. These requirements are specified using OO-Method, an automatic software production method based on transformation models. We illustrate the application of our proposal in a Car Rental real system.

Keywords: *Functional reuse, requirement specification, functional size, measurement.*

1. Introduction

Software reuse, defined as the process of using existing software artefacts instead of building them from scratch [1], is a key element in improving the way software is developed and supported over its life cycle. Potential benefits of reuse include lower maintenance costs, shorter development time and improved product quality and reliability [2] [3]. These benefits are particularly evident in new software development technologies such as Object-Oriented (OO) Systems development, where several empirical studies have verified the positive impact of reuse on productivity [4][5][6][7]. However, the majority of these studies have been carried out at the source code level, which saves effort only late in the life cycle.

Given that the Model Driven Architecture (MDA) paradigm is based on model transformation, and that such models are constructed at early stages of the life cycle, this implies a greater need for reusability at such stages. At present most practitioners ignore requirements engineering even where requirements reuse has significant potential for creating further reusability at later stages in the product life cycle [8]. In addition, there has been surprisingly little research on quantification of requirements reuse [9], [10].

We are aware of the need for software reuse from non-functional [33] and functional perspectives. However, this paper aims to automatically quantify reusable functionality from requirements specifications modelled with the OO-Method approach [11], an automatic software production method compliant with MDA principles.

In practice, quantification of reused functional requirements is vital for managers who need to analyse

reuse figures for project planning and their impact on effort and costs studies.

Our proposal is based on our previous research on Functional Size Measurement (FSM) in requirements engineering [12], [13]. Currently, of the various size measurement methods that have been proposed, four methods are considered to be standard (IFPUG FPA: ISO/IEC 20926; MARK II FPA: ISO/IEC 20968; NESMA FPA: ISO/IEC 24570; COSMIC: ISO/IEC 19761[29]). We have taken and adapted an FSM procedure compliant with COSMIC, called RmFFP [12], with the aim of quantifying functional reuse from Object Oriented requirement specifications.

In this paper, our research has focused on two basic questions: what to measure as reuse, and how to measure it. Therefore our objectives are twofold: 1) to identify what requirements model primitives could be identified as reusable, and 2) to quantify reusability identified using RmFFP.

The rest of this paper is organized as follows: In Section 2, we discuss work relating to reusability measurement. Section 3 provides a brief introduction to the OO-Method Requirements Model. In Section 4, we introduce levels of requirement reuse and a reuse indicator derived from size of functional requirements, which is achieved by means of automation of COSMIC in the OO-Method environment. Section 5 presents an illustrative example on the application of our proposal. Finally, we present some conclusions and further work.

2. Previous research

Several initiatives on reuse metrics have been proposed since the early 90s.

Bieman [14] and Karunanithi [15] defined a set of metrics for object-oriented systems from three reuse perspectives: server, client, and system. Chidamber and Kemerer[16] proposed a metrics suite for object oriented design. Among these, the most significant in relation to reuse is the Depth of Inheritance Tree metric, which calculates the depth of inheritance hierarchies. Chidamber and Kemerer assert that this metric can help managers manage reuse opportunities by measuring inheritance.

In 1995, Abran and Desharnais [17] proposed the first version of functional reuse metrics based on the FPA technique. They illustrated how these metrics could be used to take into account the benefits of reuse in a cost-benefit analysis. On the basis of this technique, Daneva [24] dealt with the identification and the measurement of reuse in the requirements conceptualization phase of the SAP R/3 component configuration cycle. Vinh Ho et al. [18], using the more recent COSMIC FSM method, proposed quantifying functional reuse based on the size of the processes referenced in the functional relationships between 'layers' . A disadvantage of this proposal is that the measurement is carried out in the design phase, where the layer concept is identified.

We have found current research on measurement of reuse at the requirements level to be scarce in the literature we have perused.

3. OO-Method Requirements Model: Basic Concepts

OO-Method is a method based on model transformations, where a requirements analysis process semi-automatically generates the primitives of a Conceptual Model, which are then converted into their associated software component counterparts through an Execution Model [11]. As the purpose of this paper is to discuss reuse that could be quantified from requirement specifications, in this section we introduce a set of complementary techniques that allow the capture of the functional properties that the system requires.

3.1. The Requirements Model

The Requirements Model [19], as shown in Figure 1, includes identification of the mission statement, construction of the Functional Refinement Tree, and the Use Case Model.

The Mission Statement is a high-level description of the nature and purpose of the system, which makes it possible to accurately determine what the system will and will not do.

The Function Refinement Tree (FRT) represents the hierarchical decomposition of the business functions of a system independently from the actual system structure. The resultant tree is merely an organization of external functions and does not say anything about the internal decomposition of the system. The leaves of this tree are use cases that represent the functions of the desired system. This tree gives an entry point for building the Use Case Model which avoids the need for starting from scratch, and helps prevent any potential confusion between the abstraction levels of Use Cases.

The Use Case Model allows us to model the system's functional requirements from the user's perspective. The leaf nodes of the Function Refinement Tree (elementary functions) are considered to be Primary Use Cases; they represent the most important functions of the system. It is also possible to have Secondary Use Cases. In this case, we have to relate the Primary Use Case to these Secondary Use Cases.

We consider a use case to be a Secondary Use Case when (1) it is duplicated in other Use Cases, and (2) when the Primary Use Cases are complex and long, and separating them helps factor the Use Case into manageable and comprehensible units.

Version 2.1 of UML [20], proposes two kinds of relationships between Use Cases: include, and extend, which are supported also by the OO-Method Requirements Model.

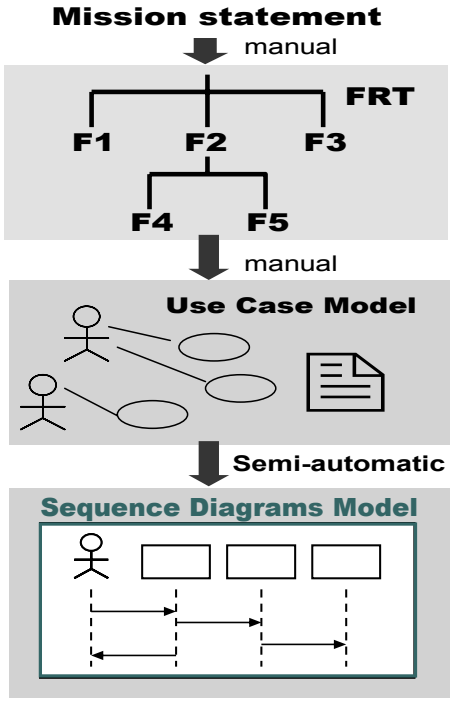

Fig 1. Requirements Process

3.2. The Sequence Diagrams Model

The Sequence Diagrams Model is built semi-automatically from use case specifications on the basis of the Metamorphosis framework that follows a linguistic engineering approach [21]. The sequence diagram model documents the internal view of a system for a Use Case. Notation is provided by UML with some extensions incorporated to classify object interactions by their nature. There are four interaction types: signal, service, query, and connect [19]. Each interaction message can also be labelled with a condition that, if satisfied, allows the interaction to occur. The syntax for this type of condition is: [Boolean-expression] message-name.

Finally, when a set of messages is implied by a particular condition or a particular iteration, it can be graphically represented by a box labelled with the condition or iteration expression. This set of messages is called Block.

4. Functional Reuse

In this section, before quantifying the reuse of requirements specified with OO-Method from a functional standpoint, we will first describe the notion of reuse level used to identify these requirements.

4.1. The notion of reuse level

For the purpose of our research, we adopt the term reuse level to mean the amount of specified functionality that a project team will reuse in the system development. Our paper borrows reuse level definitions used by the third author in her earlier studies of requirements reuse in COTS-projects [9]. Therein this author distinguished between three levels of reuse, based on (i) analysis of COTS-requirements changes [22], (ii) measurement concepts captured in a Functional Size measurement model (namely the Function Point Analysis), and (iii) modes of component reuse investigated by Karlsson [23]:

- Level 3 refers to conceptual modelling units (e.g. scenario processes and data entities) that were reused in a requirements specification without any changes. Scenarios with higher reuse rate at this level are known to have greater potential for reuse.

- Level 2 refers to minor enhancements applied to units in conceptual models. A minor enhancement is defined as a change of a certain parameter of a business scenario process or a data entity that does not result in a change of the business process logic.

- Level 1 refers to major enhancements applied to the process or data modelling units in conceptual models. A major enhancement is any considerable modification in the definition of a scenario process or a data entity that affects the business process logic from the user's point of view.

- Level 0 refers to newly introduced processes and data entities. This does not mean a reuse category; it just helps us to partition the overall requirements.

In this paper we focus on identifying and measuring functionality that is reused without any changes (Level 3) in object-oriented requirements specifications. We

deliberately do not investigate other reuse levels, as they do not fall within the scope of this paper.

4.2.The functional reuse indicator

The most commonly reported reuse indicator is the "reuse percentage", which at the beginning of the 90's was expressed as a proportion of (i) the total lines of code previously used and included in the source files, and (ii) the total lines of code in the product source files [25].

The purpose of the "Reuse Percentage" measurement is to indicate the portion of a product, product release, or organizational effort that can be attributed to reuse.

As our reuse perspective is purely functional, applying the "Functional Reuse Percentage" allowed us to derive a reuse indicator that includes Reused Functional Requirements (RFR) as a percentage of Total Functional Requirements Delivered (TFR).

$$Functional\_reuse\_percentage = \frac{RFR}{TFR} * 100\% \quad \textbf{(1)}$$

Project staff, in order to be able to use this indicator, first need to have a way to quantify the amount of reusable requirements (RFR) and the amount of all functional requirements (TFR) that are specified for development in an Object Oriented application.

The quantification of functional requirements has been most often carried out with IFPUG Function Point Analysis (FPA) [26], the first Functional Size Measurement Method developed in the 80s. However, researchers [27] indicate that, although IFPUG FPA has had a wide take-up in industry, it has only a limited applicability to modern types of software systems and new development paradigms. Due to these weaknesses, we have chosen the COSMIC Full Function Point (FFP) method [28], which is suitable for measuring any software component and can be applied to various types of software.

4.3 Obtaining the 'functional reuse percentage' with RmFFP

We first introduce COSMIC FFP and then show how it has been automated within the OO-Method approach.

The general COSMIC measurement process starts with the **measurement strategy phase** which is necessary to determine measurement scope and purpose, functional users and the level of granularity of the description of the piece of software to be measured. In contrast to traditional FSM methods –where the user concept is limited to human users– COSMIC has a broader user concept defined as anything that interacts with the software being measured.

The second phase, known as the **mapping phase**, takes as input the specifications of the Functional User Requirements (FURs) to be measured. This collection of FURs can be decomposed into a set of functional processes. Each functional process is a unique, cohesive and independently executable set of movements of data groups, with data groups being defined as a distinct, non-empty, non-ordered and non-redundant set of data attributes. The measurement method does not require the identifying of data attributes; these might be identified if a sub-unit of measure is required. The identification of the data movements of each functional process is then carried out. A data movement moves one or more data attributes that belong to one data group. The four valid types of data movement are: input, read, write and exit.

Finally, the **measurement phase** begins with the application of the measurement function to each instance of a data movement by assigning a numerical quantity, 1 CFP (COSMIC Function Point), which is the basic unit and is equivalent to a single data movement. Finally, the application of the aggregation function continues when the data movements of all the functional processes have been measured.

Taking into account this standard method [28], we have defined a measurement procedure called RmFFP [12] designed to automatically measure functional size from requirements specifications in the OO-Method context.

In order to measure functional reusability using the "reuse percentage" indicator, we have adjusted the RmFFP procedure in the following way:

With respect to the measurement strategy phase:

- Our measurement purpose is to measure the size of the total FURs of the application delivered, as well as the size of the FURs which were reused in order to obtain an indicator of functional reuse from requirement specifications.
- Measurement scope is determined by developer viewpoint, which covers all the functionality represented by the OO-Method Requirements Model.
- The granularity level is medium, because we need to know certain aspects such as the relationships between use cases.

With respect to the mapping phase: No change has been brought about in the mapping rules defined to represent the COSMIC concepts in the respective OO-Method Requirements Model primitives (See Table 1).

Table 1. COSMIC and the OO-Method Requirements Model

COSMIC Concepts [29]		Primitives of the Requirements Model [19]
Users		**Rule 1.** Actors of use case diagram
Boundary		**Rule 2.** Use case diagram
Functional processes		**Rule 3.** Primary use cases **Rule 4.** Secondary use cases
Data groups		**Rule 5.** Entity classes of sequence diagram **Rule 6.** Actors of use case diagram
Attributes data		**Rule 7.** Constant and variable arguments of the interaction messages of service type
Data movements	Entry	**Rule 8.** Signal message with value Input
	Read	**Rule 9.** Query message **Rule 10.** Condition of message **Rule 11.** Precondition of use case **Rule 12.** Condition of relation Extend **Rule 13.** Integrity constraint
	Write	**Rule 14.** Service message - New - Destroy - Update
	Exit	**Rule 15.** Signal message with value Output

With respect to the measurement phase:

- We apply the measurement function by assigning 1 CFP to each identified data movement (‘*x’*) that belongs to a functional process (P).

$$f(x) = 1CFP, \quad \forall x \in P = \{entry, exit, read, write\} \quad \textbf{(2)}$$

In order to quantify functional reusability by using the ‘reuse percentage’ indicator (See Equation 1), the aggregation functions were adjusted.

We need first to calculate two values: Reused Functional Requirements (RFR) and Total of Delivered Functional Requirements (TFR).

We explain below how each of these values was calculated.

4.3.1. Reused Functional Requirements

Firstly, we must identify what is considered as reuse in the definition of Reuse Level 3 to be quantified. Saeki proposed an approach based on patterns for reusing use cases [18], utilising the semantics of the <<extends>> and <<uses>> relationships. However, this author does not include reuse measurement concepts in their studies.

By analysing the OO-Method Use Case Model, we identify as ‘reused’ those use cases and actors which share different kinds of relationships, which are described below.

Relationships between use cases. An include relationship is intended to be used when there is common behaviour in two or more use cases. This common behaviour is then extracted to a separate use case, to be included in all the Base Use Cases having this commonality (See Figure 2). Since the primary use of the include-relationship is for reuse of common parts, what is left in a base use case is usually not complete in itself. In other words, for the base use case to be meaningful, it will require the included parts (the so-called Secondary Use Case). This is reflected in the direction of the relationship, which indicates that the Base Use Case depends on the addition of Secondary Cases Uses but not vice versa [20].

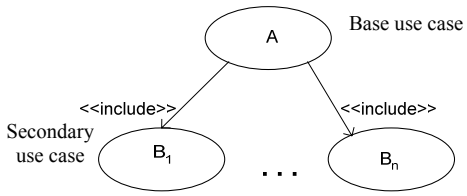

Fig 2. Include relationship in OO-Method

An *extend relationship* (See Figure 3) is intended to be used when there is some additional behaviour that

should be added (that is, a Secondary Use Case), possibly conditionally, to the behaviour defined in another use case. This additional behaviour could be reused by one or more use cases. The extending use case can access and modify the attributes of the base use case; however, the base use case is not aware of the extending use case [20].

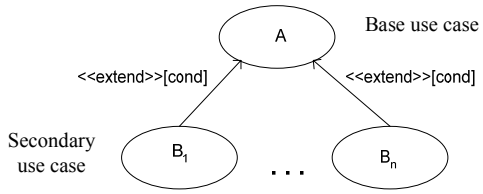

Fig 3. Extend relationship in OO-Method

The secondary use cases are potential candidates for reuse. However, we only will take into account the following situations:

1. When the secondary use case has, as minimum, two include relationships inputs (See B1 in Figure 4a).
2. When the secondary use case has, as minimum, one include relationship input and one extend relationship output (See B1 in Figure 4b).
3. When the secondary use case has two extend relationship outputs (See B1 in Figure 4c).

Relationship between actors. Another potential reuse within use-case models occurs between actors: an actor in a use-case diagram can inherit data attributes from another actor. For example, the "Foreign Student" actor inherits attributes from "Student" (i.e. name, address, and phone). A foreign student is a student, the only difference being that he or she is subjected to different rules and policies (for instance, the foreign student pays higher tuition fees than the local student).

As we can see in Table 1, the actors are identified as "data groups". According to COSMIC FFP, the identification of data groups will rely on the identification of data movements (one data movement by each different data group implied). Therefore, the actors generalized into a super-actor will be considered only as one data group, which implies that we will identify a single data movement.

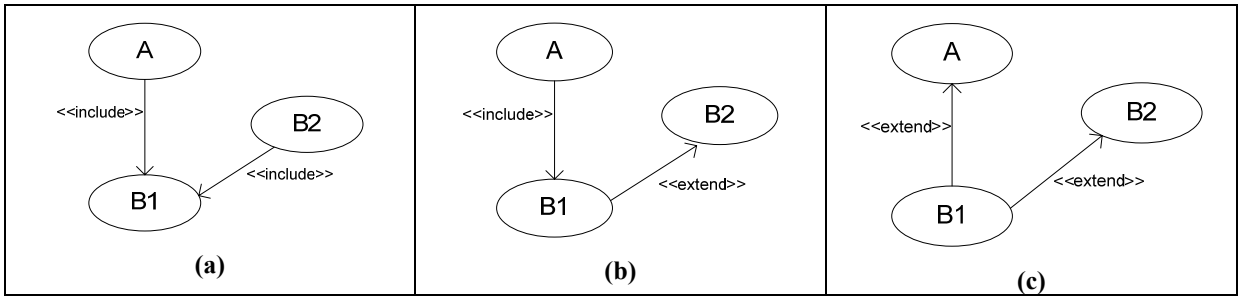

Fig 4. Situations relating to reused secondary use cases

We must now define how to measure functional reuse:

In OO-Method, as the system's functionality is specified using use cases and each use case is represented by one or more sequence diagrams, the value for this functionality is obtained by applying the following aggregation functions. As a first step, we add together all data movements identified in the sequences diagram of a (primary or secondary) use case.

$$Size(Use\_Case)_j = \sum_{i=1}^{n} Size(Data\_movement)_i \quad \textbf{(3)}$$

In order to obtain the RFR value, we will measure the functional size of the secondary use cases selected as reuse (see Equation 4).

$$RFR = \sum_{i=1}^{n} Size(Secondary\_UseCase)_i \quad \textbf{(4)}$$

4.3.2. Total of Delivered Functional Requirements (TFR).

To quantify the total FUR in a project, we work from the developer's viewpoint, from which we can deduce the entire functionality of the software that has to be delivered.

However, an additional aggregation function is defined due to the relationships that appear between use cases. Therefore the functional size of a Base Use Case (BUC) related to one or more Secondary Use Cases (SUC) is calculated by applying the following equation:

$$Size_T(BUC) = \sum_{i=1}^{n} Size(SUC_i) + Size(BUC) \quad \textbf{(5)}$$

In (5), the expression *'$Size_T$'* means the total size of the base use case.

Finally, to obtain the TFR, we add up the functional sizes of all the use cases.

$$TFR = \sum_{i=1}^{n} Size(Use\_Case)_i \quad \textbf{(6)}$$

To aid the reader's understanding of reusability measurement from a functional perspective, in the next section we will illustrate the application of adjusted RmFFP process to measure functional reuse.

5. Applying RmFFP in order to measure functional reuse

According to Jacquet and Abran [30], three steps are required to apply a measurement procedure: 1) software documentation gathering, 2) construction of the software model, and 3) application of numerical assignment rules which were adapted to measure functional reuse. Figure 4 contains a representation of these steps used in applying RmFFP.

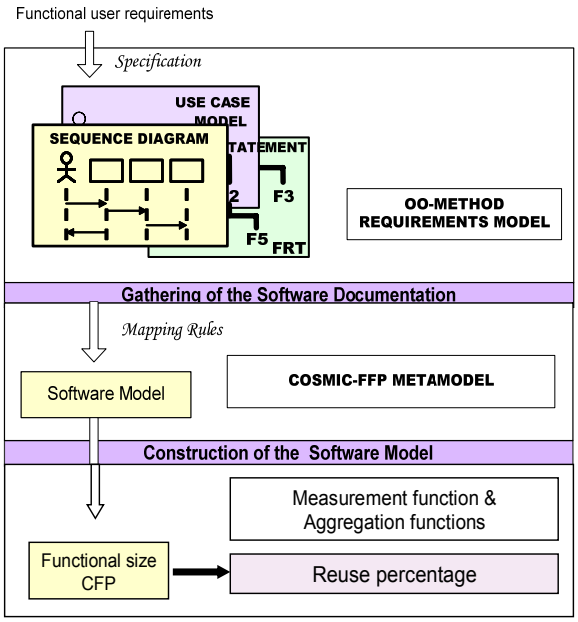

Fig 4. Application of the RmFFP procedure

The functional specification used to illustrate the application of RmFFP is that of a company that needs to automate the management of car rentals to customers.

5.1. Gathering of Software Documentation

As shown in Figure 4, functional requirements are semi-formally specified using the OO-Method requirements model. We are aware that quality of software documentation will affect measurement quality. The OO-Method approach, based on model transformation, allows our requirements specification to be traceable, consistent, unambiguous and modifiable. These quality attributes will reflect positively on accuracy since there will be a greater degree of proximity between the size estimated and the size of the final application.

As we can see in Figure 1, the OO-Method requirements model includes a mission statement, a functional refinement tree, a use case model, and a sequence diagram model. The mission of the Car Rental system is to "automate the management of cars, rentals, and customers of the company. The main activity, car rental, involves another series of derived activities such as the maintaining and repairing of cars, additional accessories to be rented (extras), and customer management. These derived activities must also be automated". Due to space limitations, we cannot fully describe the functional specification in this paper. However, the first level functional groups are shown in Figure 5 (Car Management, Customer Management, User Management and Contract Management). These functional groups are represented in the main Use Case diagram as packages.

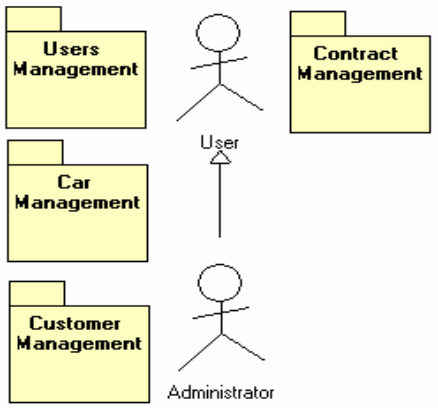

Fig 5. Use Case view of Car Rental system

Figure 6 represents the use case diagram for the functional group: Car Management.

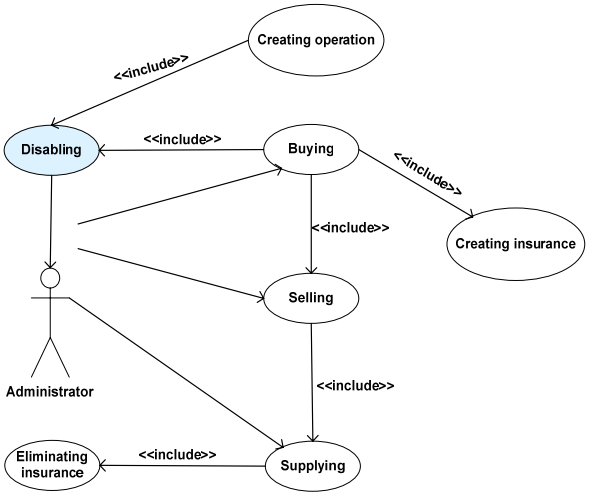

Fig 6. Use Case diagram for Car Management

5.2. Construction of the software model

This step consists of identifying the relevant elements in the OO-Method requirements specification that add to functional size, by applying the mapping rules shown in Table 1. The fundamental result of this step is a collection of identified data movements that will be quantified in the following Step (application of numerical assignment rules).

To illustrate the identification of data movements, we are going to use only the functional process "Assignment of Extras", which is represented by the Sequence Diagram shown in Figure 7. This scenario describes the addition of an extra or a set of extras to a contract.

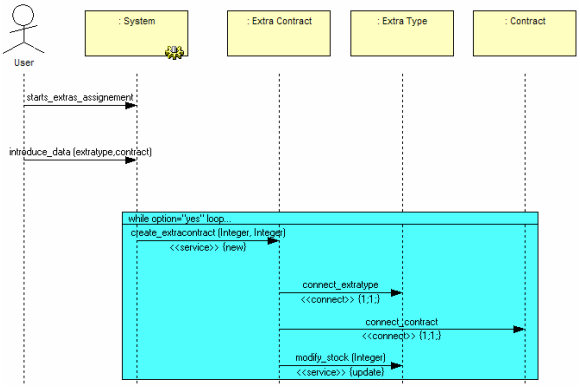

Fig 7. Sequence diagram: Assignment of Extras

This scenario starts with data input by the user, which is represented by the message "introduce_data" with the extratype and contract arguments. Applying Rule 8, this message generates three data movements because there is data input that involves arguments from three different classes (Extra Type, Contract, and Extra Contract). When the data is introduced, the system creates a new object from the ExtraContract class using the message "create_extracontract", which has the stereotype <<service/new>>. Applying Rule 14 from Table 1, this message is identified as a write data movement. This new object needs to select its corresponding Extra Type and Contract objects. In addition, the stock of an Extra Type is modified using the message modify_stock, which has the stereotype <<service/update>>. Reapplying Rule 14 (Table 1), this message is identified as a write data movement. This is an iteration that repeats until all the desired extras are included in the Contract (Extra Contract). In this case, despite the repeated execution of these messages, we consider them only once. Therefore, we have identified 3 entry data movements and 2 write data movements.

5.3. Application of numerical assignment rules

This step starts with assigning 1 CFP to each identified data movement in the 'Assignment of Extras' functional process, which is the thirtieth use case of the Car Rental System (See Table in the Appendix). We then sum up the quantified data movements by applying Equation 4, the result of which is a size of 5 CFP, as shown below:

$$Size(Assignment\_of\_Extras)_{30} = \sum_{i=1}^{5} Size(Data\_movement)_i = 5CFP$$

We carried out these actions to calculate the functional size for each of our 7 secondary use cases and each of our 28 primary use cases identified in the Car Rental system. Then, by applying Equation 6 (TFR), we obtained 124 CFP as the total of delivered functional requirements (See Appendix).

$$TFR = \sum_{i=1}^{28} Size(\text{Pr}imary\_Use\_Case)_i = 124CFP$$

As we can see in Figure 6, although there are various include relationships between use cases, only the "Disabling" use case is reused by both "Buying" and "Creating Operation" use cases. Therefore by applying the Equation 4 (RFR), 3 CFP is obtained as the total of reused functional requirements.

$$RFR = \sum_{i=1}^{1} Size(Disabling)_1 = 3CFP$$

Finally, replacing both values in the reuse percentage indicator, we obtained a figure for functional reuse of 2.42% for Level 3 (reused functionality in a requirements specification without any changes). This implies that the Car Rental System could have much

more reusable functionality. However, in this paper we have only focused on level 3 of reuse proposed by Karlsson.

$$Functional\_reuse\_percentage = \frac{RFR}{TFR} * 100\% = \frac{3}{124} * 100\% = 2.42\%$$

5. Conclusions and Future Work

This paper describes how to automatically measure reusability from a functional viewpoint. In order to do this we adjusted the RmFFP procedure, based on the COSMIC standard method, so as to quantify the amount of reused functional requirements within the OO-Method context. We consider that potential reuse can be modelled through three generalization relationships supported by the OO-Method requirements model: 1) 'Extend dependencies' between use cases; 2) 'Include dependencies' between use cases; and 3) 'Inheritance' between actors.

We have applied the RmFFP procedure to the Car Rental real system in order to facilitate understanding of the quantification process.

We plan to explore reusability throughout the entire OO-Method development process. Moreover, although reproducibility and ease of use of RmFFP has been empirically validated [31][32], we intend to carry out further empirical studies to validate this adjusted version of RmFFP; which allows quantifying functional reuse of object oriented requirement specifications.

Acknowledgments

This work has been developed with the support of Spanish Ministry of Education and Science under the project SESAMO TIN2007-62894 and by the Cooperation Agreement between Ministry of Education of Brazil and Spain.

References

[1] Kim, Y., & Stohr, E. A, Software Reuse: Survey and Research Directions. Journal of Management Information Systems, 14 (4), 113-147, 1998.

[2] Rombach H. D., Software reuse: a key to the maintenance problem. In Information and Software Technology Journal, 33(1), February 1991.

[3] Wiles E., Bott F., Eight steps to your own economic model of software reuse, Technical Report, Department of Computer Science, University of Wales, United Kingdom, available at http://www.aber.ac.uk/~edw97/work/reports/

[4] Melo W., Briand L., Basili V., Measuring the Impact of Reuse on Quality and Productivity in Object-Oriented Systems, Technical Report, Univ. of Maryland, Dep. of Computer Science, College Park, MD, USA 20742. January. 1995.

[5] Brito F., Melo W., Evaluating the Impact of Object-Oriented Design on Software Quality, In Proceedings of third International Software Metrics Symposium (METRICS'96), pp. 90-99, Berlin, Germany,1996.

[6] Tewari R., Empirical Investigation of Software Reuse in Object-Oriented Systems, The Pennsylvania State University CiteSeer Archives, March 1995.

[7] John A. Lewis , Sallie M. Henry , Dennis G. Kafura , Robert S. Schulman, An empirical study of the object-oriented paradigm and software reuse, Conference proceedings on Object-oriented programming systems, languages, and applications, p.184-196, October Phoenix, Arizona, United States, November 1991.

[8] William N. Robinson , Suzanne D. Pawlowski , Vecheslav Volkov, Requirements interaction management, ACM Computing Surveys (CSUR), v.35 n.2, p.132-190, June 2003.

[9] Daneva M., Evaluating the value-added benefits of using requirements reuse metrics in ERP projects, ACM SIGSOFT Software Engineering Notes, v.26 n.3, p.155-163, May 2001.

[10] Pastor O., Molina, J.C. Model Driven Architecture in Practice: A Software Production Environment base don Conceptual Modeling. Springer-Verlag. 2007.

[11] Condori-Fernández N., Abrahão S., Pastor O.: On the Estimation of the Functional Size of Software from Requirements Specifications. Journal of Computer Science and Technology, Springer Boston, 22(3): 358-370 (2007).

[12] Daneva M. Deriving function points from SAP business processes and business objects, Journal of Information and Software Technologies, 1999.

[13] Bieman, J. and Karunanithi, S. "Candidate reuse metrics for object oriented and Ada software." In IEEE-CS 1st International Software Metrics Symposium, 1993.

[14] Bieman, J. "Deriving measures of software reuse in object oriented systems." In BCS-FACS Workshop on Formal Aspects of Measurement in Springer-Verlag, 1992.

[15] Chidamber, S. and Kemerer, C. "A metrics suite for object oriented design." IEEE Transactions on Software Engineering 20 (6 1994): 476-493.

[16] A. Abran and J. Desharnais, ``Measurement of functional reuse in maintenance,'' Journal of Software Maintenance: Research and Practice, vol. 7, no. 4, pp. 263-277, 1995.

[17] Ho V.T., A. Abran, S. Oligny, Using COSMIC-FFP to Quantify Functional Reuse in Software Development, Proc. of the European Software Cost Estimation Conference (ESCOM-SCOPE), April 18-20, 2000.

[18] Saeki M. Patterns and Aspects for Use Cases: Reuse techniques for Use Case Description, in proceedings of the 4 th International Conference on Requirements Engineering (ICRE), 2000.

[19] Insfran E., Pastor O. and Wieringa R., "Requirements Engineering-Based Conceptual Modelling". Journal Requirements Engineering, Springer-Verlag, 2002.

[20] Object Management Group, Unified Modeling Language: Superstructure version 2.1.1 (with change bars) formal/ 2007-02-03.

[21] Diaz I., Sanchez J. y Pastor O., Metamorphosis: A framework for the functional requirements analysis, Workshop on Requirements Engineering (WER), Porto-Portugal, 2005, pp. 233-244.
[22] Keller, G., and Teufel, T. SAP R/3 Process Oriented Implementation, Addison-Wesley Longman, Harlow, 1998.
[23] Karlsson, E.-A. (ed.). Software Reuse: A Holistic Approach, John Wiley & Sons, Chichester, 1998.
[24] Daneva M., Measuring Reuse of SAP Requirements: a Model-based Approach, Proceedings of the Symposium on Software reusability, Los Angeles, California, ACM New York, USA pp. 141 – 150, 1999.
[25] Poulin, J. Measuring Software Reuse: Principles, Practices, and Economic Models, Addison-Wesley, Reading, MA, 1997.
[26] IFPUG, Function Point Counting Practices Manual Release 4.1, International Function Point Users Group, Westerville, Ohio, USA, 1999.
[27] Kitchenham B. Counterpoint: The Problem with Function Points, Status Report. IEEE Software, 1997, 14(2): 29-31.
[28] Abran A., J. M. Desharnais, S. Oligny, D. St-Pierre, and C. Symons, "COSMIC Measurement Manual – Version 3,0, The COSMIC Implementation Guide for ISO/IEC 19761:2003" École de technologie supérieure- ETS, Montreal (Canada) 2003. Available free at: http://www.gelog.etsmtl.ca/COSMIC/
[29] ISO, "ISO/IEC 19761: 2003, Software Engineering: COSMIC-A Functional Size Measurement Method", International Organization for Standardization-ISO, Geneva, 2003.
[30] Jacquet J. P. and Abran A., "From Software Metrics to Software Measurement Methods: A Process Model", in International Software Engineering Standards Symposium and Forum, ISESS 97: IEEE-Computer Society Press, 1997, pp. 128-135.
[31] Condori-Fernández N., Pastor O.: Evaluating the Productivity and Reproducibility of a Measurement Procedure. Springer Berlin / Heidelberg, ER (Workshops) 2006: 352-36.
[32] Condori-Fernández N., Pastor O.: An Empirical Study on the Likelihood of Adoption in Practice of a Size Measurement Procedure for Requirements Specification, IEEE Computer Society, QSIC 2006: 133-140
[33] Leite J.C.S, Yu Y.,Liu Y., Yu E., Quality-Based Sofware Reuse, LNCS, Springer, Advanced Information Systems Engineering, Software Quality, CAISE, June 2005, Portugal, pp 535-50.

Appendix

Table A1. Measuring the functionality of the Car Rental System

DATA MOVEMENTS		ENTRY	READ	WRITE	EXIT	Partial Size	Size of SUC	Functional Size
ID	FUNCTIONAL PROCESS							
1	Buying	2	1	1		4	22	26
2	Selling	2	1	1		4	9	
3	Delivering	2	1	2		5	4	
4	Creating rate	1	1	1		3		3
5	Eliminating rate	1	1	1		3		3
6	Modifying rate	1	1	1		3		3
7	Creating garage	1	1	1		3		3
8	Eliminating garage	1	1	1		3		3
9	Modifying garage	1	1	1		3		3
10	Creating insurance	3	1	2		6		
11	Eliminating insurance	1	1	2		4		
12	Modifying insurance	1	1	1		3		3
13	Creating insurance company	1	1	1		3		3
14	Eliminating insurance company	1		1		2		2
15	Modifying insurance company	1	1	1		3		3
16	Creating operation	3	2	1		6	3	9
17	Disabling	1	1	1		3		
18	Supplying	1	1	1		3		
19	Eliminating operation	1		1		2		2
20	Finalizing operation	1	1	1		3	3	6
21	Creating client	1	1	1		3		3
22	Eliminating client	1	1	1		3		3
23	Modifying client	1	1	1		3		3
24	Renting	3		3		6	5	11
25	Modifying contract	1	1	1		3		3
26	Returning	1	4	4	1	10		10
27	Creating type of extra	1	1	1		3		3
28	Eliminating type of extra	1		1		2		2
29	Modifying type of extra	1	1	1		3		3
30	Assigning extras	3		2		5		
31	Creating user	1	1	1		3		3
32	Eliminating user	1		1		2		2
33	Modifying user	1		1		2		2
34	Elevating	1		1		2		2
35	Dismissing	1		1		2		2
		CAPA 1						124