On the Estimation of the Functional Size of Software from Requirements Specifications

Nelly Condori-Fernández, Silvia Abrahão, and Oscar Pastor

Department of Information Systems and Computation, Valencia University of Technology, Camino de Vera s/n, 46020 Valencia, Spain

E-mail:nelly@dsic.upv.es; sabrahao@dsic.upv.es; opastor@dsic.upv.es

Received March 15, 2006; revised December 5, 2006.

Abstract This paper introduces a measurement procedure, called RmFFP, which describes a set of operations for modeling and estimating the size of object-oriented software systems from high-level specifications using the OO-Method Requirement Model. OO-Method is an automatic software production method. The contribution of this work is to systematically define a set of rules that allows estimating the functional size at an early stage of the software production process, in accordance with COSMIC-FFP. To do this, we describe the design, the application, and the analysis of the proposed measurement procedure following the steps of a process model for software measurement. We also report initial results on the evaluation of RmFFP in terms of its reproducibility.

Keywords requirement specification, product metrics, measurement technique

1 Introduction

It is generally accepted that software applications need to be well aligned with business strategies of organizations. To achieve this goal, a requirements modeling is necessary to be sure that further software components provide appropriate solutions to the system problems represented by the corresponding requirements. Nowadays, software size is a key factor in the development of budgets, effort, and schedule estimation models[1]. These models are useful to generate a variety of indicators for the entire software life cycle. Hence, the capability to accurately quantify the software size at an early stage of the development life cycle is a critical issue.

A Functional Size Measurement (FSM) method measures the size of the software by quantifying the Functional User Requirements[2]. These requirements are a sub-set of the user requirements that exclude quality requirements and any technical requirements. The most widely used FSM method is Function Point Analysis (FPA)[3], which is maintained by the International Function Point Users Group (IFPUG). It is based on the method proposed by Albrecht[4] developed to measure Management Information Systems (MIS). Although IFPUG FPA has obtained increasing popularity in the industry, its lack of applicability to all software types and the rapid evolution of the development paradigms have produced many variations to this method[5-8]. Some studies also claim that the function point definition itself has not been clarified and has generated confusion among both practitioners and academics[9,10].

In order to address these weaknesses, COSMIC-FFP has emerged as the second generation of FSM methods. This method has a generic character that is applicable to several software domains. It is compatible with modern software engineering concepts and is an international standard for FSM approved by ISO (ISO/IEC 19761[11]). However, the generality of COSMIC-FFP needs to be instantiated through a more specific and systematic procedure in accordance with a software development method. Currently, there are some approaches that apply COSMIC-FFP to estimate functional size of future software applications from high-level specifications[12,13]. However, these proposals are weak in terms of rigorous and systematic definition and validation. Important aspects such as reproducibility of the results are not guaranteed.

This paper confronts these problems directly by analyzing and adapting COSMIC-FFP to estimate the functional size of a software application from its corresponding requirements specification in the OO-Method context[14].

An FSM procedure, called RmFFP, is defined in a systematic way using the process model for software measurement suggested by Abran and Jacquet[15]. This process model comprises four phases. In the first phase, the concept to be measured is defined, and the rules to measure this concept are conceived. In the second phase, the measurement method is applied to obtain the size of the software application. In the third phase, the results provided by the measurement method are presented and verified. Finally, the results are used in different types of models.

This work includes: 1) the definition of a set of mapping rules that allows the significant primitives of the Requirements Model to be identified; 2) the definition of a set of measurement rules that allows the estimated functional size of OO systems produced using OO-Method approach to be quantified; 3) the application of both rules set to some real case studies; 4) the

Regular Paper

Supported by the Ministry of Science and Technology under the DESTINO Project, Ref. TIN2004-03534, Spain.
evaluation of these results in terms of their reproducibility. Therefore, the first three phases of the measurement process of Jacquet and Abran fall within the scope of this paper. The last phase will be dealt with in future work.

The structure of the paper is as follows. Section 2 presents an overview of the OO-Method development process emphasising the Requirements Model. Section 3 describes the design of the RmFFP procedure. Section 4 presents the application of RmFFP in a selected case study. Section 5 evaluates the reproducibility using RmFFP. Section 6 discusses the related work. Finally, Section 7 presents our conclusions and future work.

2 OW-Method Development Process

The OW-Method is a method based on model transformations, where a requirement analysis process semi-automatically generates the primitives of the Conceptual Model, which are then converted into their associated software component counterparts through the Execution Model. The software production process in OW-Method is represented by three models (see Fig. 1). These are the Requirements Model, which captures the system’s functionality; the Conceptual Model, which captures the static and dynamic properties of the functional requirements and allows the abstract specification of the user interfaces; and the Execution Model, which allows the transition from the problem space to the solution space. The software application can be generated in a systematic and automatic way on different platforms.

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 the entry point for building the Use Case Model instead of starting from scratch, and it avoids the potential problem of mixing the abstraction level of Use Cases.

Finally, the Use Case Model allows the modeling of the system’s functional requirements from the user’s perspective. The leaf nodes of the Function Refinement Tree (elemental 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. These use cases are important for organizing and managing complexity through relationships among use cases that are stereotyped as EXTEND and INCLUDE.

2.2 Requirements Analysis Process

The Requirements Analysis Process (RAP) begins when the user requirements have been represented through the Use Cases Model. Its purpose is to ensure the consistency between requirements specification and subsequent phases of the software development process (see Fig. 1).

In this process, the sequence diagram model shows the internal view of a system for a Use Case. Notation is provided by UML, with some extensions incorporated to classify the different nature of object interactions. These interaction messages are: signal, service, query, and connect.

- **Signal Messages.** They are labelled with the stereotype <<signal>> and represent interactions between an actor (external entity type) and the system. The only property for this message type is direction, which can have two types of values: input and output.
- **Service Messages.** They are labelled with the stereotype <<service>>. These messages are interactions where the receiver class represents an object that changes its state when the interaction occurs. The changes can be of three types: new, update, and destroy.
- **Query Messages.** These messages are labelled with the stereotype <<query>>. They represent queries on related objects or on a class population.
- **Connect Messages.** These messages are labelled with the stereotype <<connect>> and are used to establish a structural relation between the participant objects in the interaction.

These messages can also be labelled with a condition.