Measuring Software Functional Size: Towards an Effective Measurement of Complexity

De Tran-Cao[1]
tran@nii.ac.jp

Ghislain Levesque[2]
levesque.ghislain@uqam.ca

Alain Abran[3]
abran.alain@uqam.ca

Software Engineering Management Research Laboratory
University of Quebec in Montreal (UQAM)

Abstract

Data manipulation, or algorithmic complexity, is not taken into account adequately in any of the most popular functional size measurement methods. In this paper, we recall some well-known methods for measuring problem complexity in data manipulation and highlight the interest in arriving at a new definition of complexity. Up to now, the concept of effort has always been associated with complexity. This definition has the advantage of dissociating effort and complexity, referring instead to the characteristics and intrinsic properties of the software itself. Our objective is to propose some simple and practical approaches to the measurement of some of these characteristics which we consider particularly relevant and to incorporate them into a functional size measurement method.

1. Introduction

Software size in function points is a measure of the size of the product and can be used to evaluate and predict some aspects of the production process, like the effort, the cost and the productivity of software development. There are two main approaches to measuring software size: the a posterior approach, such as LOC [9], and the a priori approach, such as the methods based on software functionality [1, 2, 3, 4, 18, 22, 24, 25]. LOC is the simplest and the earliest method used to measure software size. While it is very useful, it has been greatly criticized [4, 11] for the way in which it defines a line of code and how it deals with different types of programming language. The a priori methods are gaining more and more attention in the software measurement community because they are independent of programming languages and they allow early estimation of the size of the end-product. When calibrated to the software environment, this provides a significant index for evaluating the development effort and assessing the cost of a software product [17]. In fact, Albrecht's Function Point Analysis (FPA) is widely used to measure the software size of management information systems (MIS). However, FPA is criticized for not taking into account complexity in an objective way [24]. Therefore, FPA is not likely to be applicable to all types of software [2].

Some attempts have been made to adapt FPA to software types which are complex in terms of data manipulation, such as real-time software, in order to objectively estimate software complexity [1, 2, 18, 27, 24, 25]. The Mark-II Function Point Method [24] is one of the approaches for doing this, however it requires historical data, which may make it difficult to apply to an application without such data. There are other FPA extensions which deal with the special characteristics of software, expressing the algorithmic difficulties or the complexity of the process of transforming (or manipulating) data from inputs to produce expected output data. Some well-known methods of this type are mentioned here. Feature Points [18] weights the algorithmic complexity based on its calculation steps and the data elements that need to be manipulated. However, no guidance is provided for identifying an algorithm at an adequate level of abstraction [2]. Another approach to taking into account the complexity of data manipulation is 3D Function Points [25]. This method measures software size in three dimensions: data, function and control (dynamic behaviors). The data dimension is measured by the number of inputs, outputs, inquiries, internal data structures and external logical files, in a similar way to the measurement of unadjusted function points of FPA. The function dimension is measured by the number of transformations, which is the sum of the number of processing steps and the number of semantic statements.

[1] PhD student, Cognitive and Computer Sciences, Université du Québec à Montréal
[2] Professor of Software Engineering, Université du Québec à Montréal
[3] Professor of Software Engineering, École de Technologie Supérieure, Montréal