Chapter 16

On the Cognitive Complexity of Software and its Quantification and Formal Measurement

Yingxu Wang
University of Calgary, Canada

ABSTRACT

The quantification and measurement of functional complexity of software are a persistent problem in software engineering. Measurement models of software complexities have been studied in two facets in computing and software engineering, where the former is machine-oriented in the small; while the latter is human-oriented in the large. The cognitive complexity of software presented in this paper is a new measurement for cross-platform analysis of complexities, functional sizes, and cognition efforts of software code and specifications in the phases of design, implementation, and maintenance in software engineering. This paper reveals that the cognitive complexity of software is a product of its architectural and operational complexities on the basis of deductive semantics. A set of ten Basic Control Structures (BCS's) are elicited from software architectural and behavioral modeling and specifications. The cognitive weights of the BCS's are derived and calibrated via a series of psychological experiments. Based on this work, the cognitive complexity of software systems can be rigorously and accurately measured and analyzed. Comparative case studies demonstrate that the cognitive complexity is highly distinguishable for software functional complexity and size measurement in software engineering.

INTRODUCTION

One of the central problems in software engineering is the inherited complexity. The quantification and measurement of functional complexity of software systems have been a persistent fundamental problem in software engineering (Hartmanis and Stearns, 1965; Basili, 1980; Kearney et al., 1986; Melton, 1996; Fenton and Pfleeger, 1998; Lewis and Papadimitriou, 1998; Wang, 2003b, 2007a). The taxonomy of the complexity and size measures of software can be classified into the categories of computational complexity (time and space) (Hartmanis, 1994; McDermid, 1991),

DOI: 10.4018/978-1-4666-0261-8.ch016

symbolic complexity (Lines of Code (LOC)) (Halstead, 1977; Albrecht and Gaffney, 1983; McDermid, 1991), structural complexity (control flow, cyclomatic) (McCabe, 1976; Zuse, 1977), functional complexity (function points, cognitive complexity) (Albrecht, 1979; Wang, 2007a; Wang and Shao, 2003).

The most simple and intuitive measure of software complexity is the symbolic complexity, which is conventionally adopted as a measure in term of Lines of Code (LOC) (Halstead, 1977; Albrecht and Gaffney, 1983; McDermid, 1991). However, the functional complexity of software is so intricate and non-linear, which is too hard to be measured or even estimated in LOC. In order to improve the accuracy and measurability, McCabe proposed the cyclomatic complexity measure (McCabe, 1976) based on Euler's theorem (Lipschutz and Lipson, 1997) in the category of structural complexity. However, it only considered the internal loop architectures of software systems without taking into account of the throughput of the system in terms of data objects and many other important internal architectures such as the sequential, branch, and embedded constructs. Because the linear blocks of code are oversimplified as one unit as in graph theory, the cyclomatic complexity is not sensitive to linear structures and external data complexity as well as their impact on the basic structures. Albrecht (1979) introduced the concept of *function point* of software (Albrecht, 1979), which is a weighted product of a set of functional characteristics of software systems. However, the physical meaning of a unit function point is not rigorously modeled except a wide range of empirical studies. The *cognitive complexity* of software systems is introduced as a measure for the functional complexity in both software design and comprehension, which consists of the architectural and operational complexities. The cognitive complexity provides a novel and profound approach to explain and measure the functional complexity of software as well as the effort in software design and comprehension. The new approach perceives software functional complexity as a measure of cognitive complexity for human creative artifacts, which considers the effect of both internal structures of software and the I/O data objects under processing (Wang, 2007a; Wang and Shaw, 2003).

This paper presents a cognitive functional complexity of software as well as its mathematical models and formal measurement. The taxonomy and related work of software complexity measurement are explored systematically. A generic mathematical model of programs is created and the relative cognitive weights of fundamental software structures known as the Basic Control Structures (BCS's) are empirically calibrated based on a series of psychological experiments. The cognitive complexity of software systems is formally modeled as a product of the architectural and operational complexities of software. A set of comparative case studies is presented on applications of cognitive complexity in software engineering, which leads to a series of important findings on the basic properties of software cognitive complexity and its quantification and measurement.

TAXONOMY OF SOFTWARE COMPLEXITIES IN COMPUTING AND SOFTWARE ENGINEERING

The measurement models of software complexities have been studied in two facets in computing in the small and software engineering in the large. The orientation of software engineering complexity theories puts emphases on the problems of *functional complexity* that are human *cognition time* and *workload* oriented. While the computational complexity theories are focused on the problems of *high throughput complexity* that are computing *time efficiency* centered. In other words, software engineering measures system-level complexities, while computational science measures algorithmic complexities.