Chapter 13
Empirical Studies on the Functional Complexity of Software in Large-Scale Software Systems

Yingxu Wang
University of Calgary, Canada

Vincent Chiew
University of Calgary, Canada

ABSTRACT

Functional complexity is one of the most fundamental properties of software because almost all other software attributes and properties such as functional size, development effort, costs, quality, and project duration are highly dependent on it. The functional complexity of software is a macro-scope problem concerning the semantic properties of software and human cognitive complexity towards a given software system; while the computational complexity is a micro-scope problem concerning algorithmic analyses towards machine throughput and time/space efficiency. This paper presents an empirical study on the functional complexity of software known as cognitive complexity based on large-scale samples using a Software Cognitive Complexity Analysis Tool (SCCAT). Empirical data are obtained with SCCAT on 7,531 programs and five formally specified software systems. The theoretical foundation of software functional complexity is introduced and the metric of software cognitive complexity is formally modeled. The functional complexities of a large-scale software system and the air traffic control systems (ATCS) are rigorously analyzed. A novel approach to represent software functional complexities and their distributions in software systems is developed. The nature of functional complexity of software in software engineering is rigorously explained. The relationship between the symbolic and functional complexities of software is quantitatively analyzed.

DOI: 10.4018/978-1-4666-2651-5.ch013

Copyright © 2013, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.
1. INTRODUCTION

Although computational complexity of algorithms and programs have been well studied in computer science (Hartmanis & Stearns, 1965; Hartmanis, 1994; Zuse, 1997; Wang, 2009), the functional complexity of software in software engineering (McDermid, 1991; Wang, 2009a) is yet to be rigorously explored from both theoretical and empirical aspects. If it is perceived that the computational complexity is a micro-scope problem concerning algorithmic analyses towards machine’s throughput and efficiency, the functional complexity of software is a macro-scope problem concerning the semantic space of software and human cognitive complexity towards a given software system (Wang, 2003a, 2007b; Wang & Chiew, 2010; Wang et al., 2006). Conventional computational complexity theories are mainly focused on time and space properties of a given problem, which is usually a function of the input size $O(f(n))$ in the domains of real number ($\mathbb{R}$). However, software functional complexity is a two dimensional hyper-structure ($\mathbb{H}_S$) between the interactions of the architectural data objects and the behavioral operations (Shao & Wang, 2003; Wang, 2009). Therefore, there is a practical need to study the functional properties of software and how they fundamentally affect human cognition, design, and manipulation in software engineering.

It is recognized that functional complexity is one of the most fundamental properties of software, because almost all other software properties and attributes such as functional size, development effort, costs, quality, and project duration, are highly dependent on it. The quantification and measurement of software functional complexity have been a persistent fundamental problem in software engineering (Hartmanis & Stearns, 1965; Basili, 1980; Kearney et al., 1986; Melton, 1996; Fenton & Pfleeger, 1998; Lewis & 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), symbolic complexity (Lines of Code (LOC)) (Halstead, 1977; Albrecht & Gaffney, 1983; McDermid, 1991), structural complexity (control flow, cyclomatic) (McCabe, 1976; Zuse, 1977), functional complexity (function points, cognitive complexity) (Albrecht, 1979; Wang, 2007a, 2009; Shao & Wang, 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 & 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 & 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 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 defined instead of various empirical studies.

In order to improve the understanding of the nature of software functional complexity, the semantic properties of software have to be systematically studied (Wang, 2006, 2009, 2001). The two-dimensional semantic space of software indicates that the cognitive complexity of software is a measure for the functional complexity for both software design and comprehension, which