Object-Oriented Cognitive Complexity Measures: An Analysis

CHAPTER · JANUARY 2014

DOI: 10.4018/978-1-4666-6359-6.ch006

2 AUTHORS:

Sanjay Misra

Covenant University Ota Ogun State, Nigeria

Adewole Adewumi

Covenant University Ota Ogun State, Nigeria

Chapter 6
Object-Oriented Cognitive Complexity Measures: An Analysis

Sanjay Misra
Covenant University, Nigeria

Adewole Adewumi
Covenant University, Nigeria

ABSTRACT

This chapter presents the analysis of ten recently proposed object-oriented metrics based on cognitive informatics. The metrics based on cognitive informatics use cognitive weight. Cognitive weight is the representation of the understandability of the piece of software that evaluates the difficulty experienced in comprehending and/or performing the piece of software. Development of metrics based on Cognitive Informatics (CI) is a new area of research, and from this point of view, for the analysis of these metrics, it is important to know their acceptability from other existing evaluation and validation criteria. This chapter presents a critical review on existing object-oriented cognitive complexity measures. In addition, a comparative study based on some selected attributes is presented.

INTRODUCTION

Software metrics play a crucial role in the process of software development. Among other things, they assist the developer in assuring quality in software. Developers utilize metrics in the different phases that make up the life cycle of software development to better understand and assess the quality of systems they have built. Developing an absolute measure is a non-trivial activity as observed in literature (Fenton, 1994). Software engineers instead, attempt to derive a set of indirect measures that lead to metrics that provide an indication of quality of some representation of software. The quality objectives may include maintainability, reliability, performance, and availability (Somerville, 2001) which are all closely related to software complexity. Software complexity is defined as “the degree to which a system or component has a design or implementation that is difficult to understand and verify” (IEEE, 1990). Software complexity can be grouped

DOI: 10.4018/978-1-4666-6359-6.ch006

into two namely: computational and psychological complexities (Fenton, 1997). Computational complexity refers to the complexity of algorithms and also evaluates the time and memory requirements for executing a program. Psychological complexity refers to the cognitive complexity. This focuses on evaluating the human effort required to perform a software task. There are several definitions of cognitive complexity. For instance, Henderson-Sellers (1996) defines cognitive complexity as 'referring to those characteristics of software that affect the level of resources used by a person performing a given task on it.' Fenton (1997) defines cognitive complexity as the measure of effort required to understand software. Zuse (1998) defines it as the difficulty of maintaining, changing and understanding software. At this point, it is worth noting that metrics and measures are often used interchangeably in software engineering. This is due to the fact that both terms have approximately similar definitions. Pressman (2001) explains 'measure' in software engineering context as 'one that provides a quantitative indication of the extent, amount, dimension, capacity, or size of some attributes of a product or process'. A metric is defined by IEEE as 'a quantitative measure of the degree to which a system, component, or process possesses a given attribute''.

Cognitive informatics (CI), is an emerging area of research that is multidisciplinary in nature. It includes researches in the field of cognitive science, computer science, informatics, mathematics, neurobiology, physiology, psychology and software engineering (Wang, 2002, 2004, 2005, 2006, 2007, 2009). The importance of the CI research is the fact that, it attempts to address the common problems of two related areas in a bi-directional and multidisciplinary approach (Wang, 2004). CI utilizes the computing technique to solve the problem of cognitive science, neurobiology, psychology, and physiology and on the other hand uses the theories of cognitive science, neurobiology, psychology, and physiology to investigate the issues in informatics, computing, and software engineering as well as their solution. For instance, measurement in software engineering is still evolving and needs a lot of efforts to standardize it (i.e. the measurement techniques for software engineering). In the last few years, a number of researchers have tried to address these problems by combining the principles of cognitive science and measurement in software engineering. The number of proposals of object-oriented cognitive complexity measures (Kushwaha & Misra, 2006; Misra & Akman, 2008; Arockiam et al., 2009; Misra et al., 2011; Misra & Cafer, 2011; Arockiam & Aloysius, 2011; Misra et al., 2012; Aloysius & Arockiam, 2012) are the results of these efforts.

Cognitive Complexity refers to the human effort required to perform a task or the difficulty experienced in understanding a piece of code or the information packed in it (Misra & Kushvaha, 2006). Understandability of code is also referred to as program comprehension and is a cognitive process that relates to cognitive complexity. In other words, cognitive complexity is the mental burden on the user who deals with the code, for example the developer, tester and maintenance staff. Cognitive complexity provides valuable information for the design of systems. High cognitive complexity indicates poor design, which sometimes can be unmanageable (Briand, Bunse & Daly, 2001). In such cases, the maintenance effort increases significantly. In this respect, cognitive complexities are important in evaluating the performance of software system; they refer to those characteristics of software which affect the level of resources used by a person performing a given task on it (Zuse, 1998). A system with reduced cognitive complexity will not only improve the quality of the code but also reduce the future comprehension as well as maintenance efforts.

The objective of this chapter is to review and compare all the available object-oriented