Chapter 3

Analyzing Growth Trends of Reusable Software Components

Kuljit Kaur
Guru Nanak Dev University, India

ABSTRACT

Reusable software components are the software modules that can be (re)used across a number of applications in a particular domain. Component users prefer to use those components which can be adapted easily for their changing requirements. So components have to evolve continuously in order to attract users. This chapter focuses on the evolutionary aspects of software components. It mentions various techniques for monitoring software evolution. It uses metrics based analysis as the technique to show software evolution of 15 reusable components from the point of view of their size, complexity, and functionality. The evolution analysis is motivated by laws of software evolution which suggest that as software ages, it increases in size and complexity (unless it is managed) and it has to offer increased functionality to please its users. The findings of the study indicate that the size of the software components (in this data set) grows at a linear rate, and complexity is well managed. However, increase in functionality is sub linear for almost all the components. It remains almost constant for some of them.

INTRODUCTION

Software crisis is characterized by two major phenomena: Lack of ability to produce software within budget and time constraints, and lack of quality in produced software (Kim & Blodyreff, 1996). In 1968, McIlroy suggested software reuse as a means for overcoming software crisis (McIlroy, 1968). He pointed towards the effective use of reusable software components to build large reliable software systems in a controlled and cost effective way. It has been observed in several studies (Melo et al., 1995; Mohagheghi et al., 2004; Mohagheghi & Conradi, 2007) that using reusable software components can improve the reliability of a system as they generally contain fewer bugs.
Component developers develop software components keeping in mind their reuse value across product lines and organizations. These reusable components are reused as is or are adapted to meet the requirements of a different project in a context other than the one anticipated during their development (Ravichandran & Rothenberger, 2003; Yu et al., 2009). They need to satisfy not only the initial demands of their stakeholders, but also need to offer support for future, changing requirements. This study focuses on the evolutionary aspects of software components. It considers the software components modeled using the object oriented approach.

Software evolution is concerned with the aging process of source code. Lehman et al. (1997) established the laws of software evolution, which suggest that as software ages, it increases in size and complexity (unless it is managed) and it has to offer increased functionality to please its users (Godfrey & German, 2008; Lehman, 1980; Lehman et al., 1997). Several studies have been done to prove or refute these laws (Godfrey and Tu, 2000; Xie et al., 2009). Godfrey and Tu (2000) found in an experiment that the Linux source code grows super-linearly in size and complexity. This research analyzes evolution of reusable software components in terms of their size, complexity, and functionality to understand their growth as they evolve. Reuse based software systems follow a different kind of development model. As they are subject to different kinds of evolutionary pressures in comparison to monolithic single use system, they are expected to have a different kind of evolutionary behaviour.

REUSABLE SOFTWARE COMPONENTS

Component Based Development (CBD) has emerged as an important paradigm for software development. In this paradigm, a software system is developed as a composite of sub-parts, rather than a monolithic entity. These sub-parts are pre-built software units, or components. A new software system for a specific domain is just assembled using the domain-specific pre-built components. This approach reduces production cost, gives a shorter time-to-market, and results in a high quality product (Mohagheghi & Conradi, 2007), the sought after goals of the software industry since long.

Reusable components are just modules that have been designed to be useful in solving all the problems including the ones which are not anticipated beforehand. Software development organizations can reuse software components which are built in-house or acquired from third parties. Third party components fall in two categories – Open Source Software (OSS) Components and Commercial Off the Shelf (COTS) components.

A widely accepted definition of a software component is given by Szyperski [3]:

A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties.

The definition covers the characteristic properties of components. It has a technical part with aspects such as independence, contractual interfaces, and composition. Explicit context dependencies refer to the component needs that it has to mention explicitly in the context of its composition and deployment. Interfaces are the means by which components connect. Each component implements two interfaces: a required interface, and a provided interface. A provided interface is a set of operations that clients can invoke. A required interface is a set of operations that a component, as a client, invokes from the provided interfaces of its servers. The definition refers to the market aspects such as composition of components by third parties.

In the context of object oriented paradigm, Valerio et al. (2001) define a component as: