Chapter 3
Assessing Modularity in Java Programs

Jorge Manjarrez Sanchez
Research Center in Mathematics (CIMAT), Mexico

Victor Navarro Belmonte
Research Center in Mathematics (CIMAT), Mexico

ABSTRACT

The main goal of a good software design is to achieve high modularity in order to promote reusability, maintainability, and reduce costs. Coupling and cohesion are two among the many measures that quantify the degree of modularity of a system. An ideal modular design is highly cohesive and lowly coupled. This chapter presents a review of some important metrics to numerically quantify coupling and cohesion and the assessment of some tools to automate their calculation in medium and large Java programs.

1. INTRODUCTION

The objective of the software business is to create good software developed in time, within budget and that comply with requirements. In order to achieve this, the software should be easy to understand, maintain, test, evolve and manage. This motivates to elicit a modular design: the whole is divided in small independent modules of functionality; hence reducing the system's complexity and each module could be developed and managed independently by collaborating teams, perhaps geographically distributed. Moreover, these modules can be reused in new systems, reducing development time and costs, or can be replaced by more efficient ones without introducing bugs, changing operations or lines of code, at least ideally. However, such an ideal state does not exist and system parts are tied because of their interactions.

The importance of good quality software for the business is indicated in the Measurement and Analysis process of CMMI Level 2. It states among the measurement objectives (The CMMi easy button, 2012) that it:

DOI: 10.4018/978-1-4666-5182-1.ch003

- Reduces time to delivery.
- Reduces total lifecycle cost.
- Delivers specified functionality completely.
- Improves prior levels of quality.
- Improves prior customer satisfaction ratings.
- Maintains and improves the acquirer/supplier relationships.

Reusability and all the other desirable properties of good software can be attained with modules due to their well-defined boundaries, which are established by a lean interface that specifies a set of responsibilities and capabilities. Having a lean interface means the module has a small number of duties and hence increases its flexibility. One of the first persons to point out this idea was Parnas (1972) with his information-hidden principle that established the basis for encapsulation, one of the fundamental concepts in object-oriented programming and other paradigms, such as Aspect Oriented Programming (Tarr, 1999).

Unfortunately, just saying a system is good is not enough; one must state objectively the degree of goodness and in terms of what. A system is good if it has a good level of modularity measured in terms of coupling and cohesion, and to express it numerically one must compute some metrics of a software system with some known tools and compare them with their manually obtained baseline. A system is said to be good if it has low coupling between components and high cohesion inside a component.

The metrics for coupling and cohesion can be calculated manually by an engineer who understands the corresponding formulas and definitions in the literature on this subject, but the size of the system matters. A small system can be manually analyzed but a system bigger than a hundred classes makes this task hard, error prone and not achievable in a reasonable time. To ease the calculation of coupling and cohesion metrics for big projects, some tools have been developed. However, these tools do not always provide accurate results; even for a human being it is hard enough to apply the concepts implied by a metric. Many modularization principles are implemented by language constructs created after the proposal of the metrics and thus they do not have a precise corresponding variable to denote them; or sometimes the metrics are conceived for a different programming language. Hence, this is an active research subject.

In this chapter, we first review some fundamental concepts to evaluate modularity by means of coupling and cohesion. Then we review some fundamental metrics and, based on these metrics, we perform an empirical evaluation to assess the quality of a system manually and by using some automated tools. This study will also allow the reader to understand the difficulty even for these tools to correctly evaluate the parameters involved in quantifying coupling and cohesion.

2. BASIC CONCEPTS

A recurring problem in software systems is that, as it increases in size and functionality, it becomes harder to code, test and maintain. Even changing a small part can be a hard task in time and cost due to the system's complexity. To avoid this problem and keep the system simple, we must design a system with two concepts in mind: coupling and cohesion.

2.1 Coupling

Software coupling indicates the number of relationships between system components. This relationship can be by association or containments. The more dependencies a component has, the more susceptible it is to change because changes in the other components it has relationships with can trigger changes in its inputs and outputs. Also, if many components depend on one component, it acquires too many responsibilities and so it is very