Title: An Evolutionary Approach to Identify Logical Components

Author: Seyed Mohammad Hossein Hasheminejad Saeed Jalili

PII: S0164-1212(14)00122-8
DOI: http://dx.doi.org/doi:10.1016/j.jss.2014.05.033
Reference: JSS 9335

To appear in:

Received date: 3-6-2013
Revised date: 2-4-2014
Accepted date: 13-5-2014

Please cite this article as: Hasheminejad, S.M.H.,An Evolutionary Approach to Identify Logical Components, *The Journal of Systems and Software* (2014), http://dx.doi.org/10.1016/j.jss.2014.05.033

An Evolutionary Approach to Identify Logical Components

Seyed Mohammad Hossein Hasheminejad, Saeed Jalili

Department of Computer Engineering, Tarbiat Modares University, Tehran, Iran
E-mail: {SMH.Hasheminejad ,Sjalili}@Modares.ac.ir

Abstract ***Context:*** Identifying suitable components during the software design phase is an important way to obtain more maintainable software. Many methods including Graph Partitioning, Clustering-based, CRUD-based, and FCA-based methods have been proposed to identify components at an early stage of software design. However, most of these methods use classical clustering techniques, which rely on expert judgment.

Objective: In this paper, we propose a novel method for component identification, called SBLCI (*Search-Based Logical Component Identification*), which is based on GA (*Genetic Algorithm*), and complies with an iterative scheme to obtain logical components.

Method: SBLCI identifies logical components of a system from its analysis models using a customized GA, which considers cohesion and coupling metrics as its fitness function, and has four novel guided GA operators based on the cohesive component concept. In addition, SBLCI has an iterative scheme in which it initially identifies high-level components in the first iteration. Then, in the next iterations, it identifies low-level sub-components for each identified component in previous iterations.

Results: We evaluated the effectiveness of SBLCI with three real-world cases. Results revealed that SBLCI is a better alternative for identifying logical components and sub-components in comparison with existing component identification methods.

Keywords Logical Component Identification, Search-Based Software Design, Genetic Algorithm

1 Introduction

Component-Based Software Development (CBSD) is a well-known methodology, which increases reusability of systems, rapidly assembles reliable systems, and efficiently reduces development cost and maintenance overhead. In the CBSD lifecycle, the most important question is how to identify logical components. According to different studies (Birkmeier,Overhage 2012; Birkmeier,Overhage 2009; Cui,Chae 2011), component identification can be divided into business component identification, logical component identification, and reverse component identification. Business component identification begins with business model and identifies business components (Wang et al. 2005; Birkmeier,Overhage 2009; Birkmeier,Overhage 2012). In fact, a business component is the software implementation of an autonomous business concept or business process, and associates with domain concepts of an information system. It consists of all the software artifacts necessary to represent, implement, and deploy a given business concept as an autonomous, reusable element of a larger distributed information system (Herzum,Sims 2000). Logical component identification intends to discover software components representing requirements except for technologies and environments to provide the starting point for designing software architecture (Kim et al. 2008). Note that a business component is more general than a logical component and in implementation, it usually contains a set of logical components. Reverse component identification refers to the situation in which software systems already exist and start from source code of a legacy system to identify source code-level components (Cui,Chae 2011).

To construct software architecture for a system, a software architect must design the software architecture in several distinct views such as 4+1 viewpoints of Kruchten (Kruchten 1995). The views are: 1) *Use case View* concerned with a small set of use cases to describe software architecture, 2) *Logical View* concerned functionalities provided by the system to end-users, 3) *Process View* concerned with dynamic aspects of the system, 4) *Development View* concerned with software management and implementation, and 5) *Physical View* concerned with a topology of software components on a physical layer. To achieve these viewpoints for software architecture, the software architect should identify logical components and communications between them. After that, it is necessary to identify sub-components of each component obtained, because for designing and modeling software architecture viewpoints like *Physical View*, i.e., deployment view, components are usually divided into several sub-components. For example, some sub-components belonging to a component must be deployed in client side and other sub-components must be deployed in server side. Another example for dividing components into sub-components is that for achieving more performance, in which some sub-components belonging to a component should be synchronized and run in