Chapter IX

An Open Architecture for Visual Reverse Engineering

Alexandru C. Telea, Eindhoven University of Technology, The Netherlands

ABSTRACT

Tool support for program understanding becomes increasingly important in the software evolution cycle, and it has become an integral part of managing systems evolution and maintenance. Using interactive visual tools for getting insight into large evolving legacy information systems has gained popularity. Although several such tools exist, few of them have the flexibility and retargetability needed for easy deployment outside the contexts they were initially built for. The lack of flexibility and limitations for customizability is a management as well as a technical problem in software evolution and maintenance. This chapter discusses the requirements of an open architecture for software visualization tools, implementation details of such an architecture, and examples using some specific software system analysis cases. The focus is primarily on reverse engineering, although the proposed tool architecture is equally applicable to forward engineering activities. This material serves the software architects and system managers as well as the tool designers.

INTRODUCTION

Businesses of many organizations heavily depend on effective maintenance of increasingly aging software. As software ages, the task of managing to maintain it becomes more complex and more expensive. Poor design, unstructured programming methods, and crisis-driven maintenance can contribute to poor code quality, which in turn affects understanding of the system properties. Program understanding (Tilley,
1998; Muller et al., 1993; Tilley et al., 1998) is a relatively young and evolving field concerned with identifying artifacts and their relationships and understanding their structure and semantics. The essence of this process is essentially pattern matching at different abstraction levels. These levels induce in turn different representations of the candidate system. Overall, the aim is to aggregate these artifacts in a hierarchical representation in order to achieve a more refined and abstract understanding of the original system. Low-level representations, such as call or module dependency graphs, help the developers to grasp the system properties. More abstract and higher-level representations, such as simplified functional, task, or architectural diagrams may be used by the management to succinctly overview the status and evolution of a given software project.

Program understanding uses several information sources, such as direct source code examination, leveraging corporate knowledge, and computer-assisted methods. In this chapter, we focus on reverse engineering (RE) methods that address the process of understanding existing (large) software systems. However, note that the analysis and results presented in this work are also useful for the forward engineering activity.

Furthermore, we shall focus on computer-assisted RE methods, which have a number of important advantages. Firstly, they represent a deterministic representation of a software system, as compared to subjective interpretations. Secondly, they are used to analyze large systems, whereas direct source code examination fails for systems larger than approximately 50,000 lines of code (Stasko et al., 1998). Thirdly, they require, in virtually all cases, less time to learn and apply. Finally, automated methods are the only ones applicable in the vast majority of the cases, given the size of the systems at hand. Managing the evolution of large software systems thus requires automated support for their understanding, which implies, at some point, the need for flexible RE tools.

Reverse engineering provides a conceptual framework for describing the process of software understanding and conceptual abstraction. This framework is supported by several RE tools. In the recent past, an impressive number of such RE tools has emerged. However, finding the “right” tool for a given application domain remains a challenging problem. This is mainly due to the fact that application systems vary from systems to systems, and thus may spawn different, often divergent requirements.

Given the above, practitioners in the RE field are left with two main choices: either pick one of the available RE tools and adapt it to one’s specific data and requirements or create a new RE tool from scratch. In most cases, the solution of choice falls somewhere between the above two scenarios. If tool adaptation or design is required, it is thus of great importance for the RE practitioner to:

- understand the often subtle trade-offs the existing tools make in their implementation,
- be able to predict the limitations before adopting a given tool,
- avail a framework for designing a customized RE tool, in case adapting an existing one is too difficult for a particular application.

Overall, these often require a detailed analysis of the architecture of the RE tools. Based on such an analysis, the RE practitioner can compare different tools to a set of requirements, estimate the customizability of a tool of choice, or estimate the effort and way to design a custom RE tool. In absence of this analysis, tool evaluation is a time-
Enterprise Information Systems for Business Integration in Global International Cooperations of Collaborating Small and Medium Sized Organisations
[www.igi-global.com/chapter/enterprise-information-systems-business-integration/44155?camid=4v1a](www.igi-global.com/chapter/enterprise-information-systems-business-integration/44155?camid=4v1a)

Application Service Provision: A Working Tool for Inter-Organizational Systems in the Internet Age
[www.igi-global.com/chapter/application-service-provision/24489?camid=4v1a](www.igi-global.com/chapter/application-service-provision/24489?camid=4v1a)