Chapter 41
Model-Driven Reverse Engineering of Open Source Systems

Ricardo Perez-Castillo
University of Castilla-La Mancha, Spain

Mario Piattini
University of Castilla-La Mancha, Spain

ABSTRACT

Open source software systems have poor or inexistent documentation and contributors are often scattered or missing. The reuse-based composition and maintenance of open source software systems therefore implies that program comprehension becomes a critical activity if all the embedded behavior is to be preserved. Program comprehension has traditionally been addressed by reverse engineering techniques which retrieve system design models such as class diagrams. These abstract representations provide a key artifact during migration or evolution. However, this method may retrieve large complex class diagrams which do not ensure a suitable program comprehension. This chapter attempts to improve program comprehension by providing a model-driven reverse engineering technique with which to obtain business processes models that can be used in combination with system design models such as class diagrams. The advantage of this approach is that business processes provide a simple system viewpoint at a higher abstraction level and filter out particular technical details related to source code. The technique is fully developed and tool-supported within an R&D project about global software development in which collaborate two universities and five companies. The automation of the approach facilitates its validation and transference through an industrial case study involving two open source systems.

INTRODUCTION

Production and distribution models of software industry have been transformed by the open source initiative (Open Source Initiative, 2011). While several commercial software companies produce and distribute software in a centralized way, the open source model advocates developing software in peer production by bartering and collaboration (Raymond, 1999).

DOI: 10.4018/978-1-5225-3923-0.ch041

The main advantage of open source code is that it maximizes the reuse of software and reduces development efforts and cost regarding software access. From an economical viewpoint, the open source model consequently allows companies to save a lot of money (Glass, 2004).

The open source's advantages encourage many companies to use open source code. Some software development companies employ open source code as a basis for developing new systems. Other companies offer maintenance support for open source systems. However, when developers or maintainers are faced with open source code, they can find some program comprehension difficulties, which prevent agility in companies (Kotlarsky, Oshri, Kumar, & Hillegersberg, 2008). These problems are owing to the team-cross and distributed development nature of open source code (Rigby, German, & Storey, 2008). This nature implies a poor, confuse (or even inexistent) documentation and there could be not many expert people since a source code system is usually maintained for many different people throughout its lifecycle (Costa, Santana, & Souza, 2009). Program comprehension is, therefore, extremely needed when maintainers try to use open source code (even more than non-open source software systems).

Program comprehension is a key reverse engineering activity which automates the analysis of the behavior of existing software systems (Canfora, Di Penta, & Cerulo, 2011; Maletic & Marcus, 2001). This activity is so important because it allows knowing all the meaningful information to be effectively used in the next reengineering stages (i.e., restructuring and forward engineering), which is aimed at migrating or evolving the existing software system.

There is a wide variety of program comprehension techniques which are categorized in two approaches: the static and dynamic analysis (T. Eisenbarth, Koschke, & Simon, 2001). Static analysis is based on the compiler theory. These techniques syntactically analyze source code to recover structural elements (e.g., the system design based on class diagrams) or to obtain some metrics (e.g., number of lines of source code, the cyclomatic complexity, the number of coupling methods, etc.). Moreover, dynamic analysis focuses on the behavior of the system derived by its execution. This kind of techniques retrieves dead code parts, detects execution bottlenecks, etc.

Traditional program comprehension techniques, however have some limitations. Firstly, traceability between high-level representations and existing source code is error-prone, which makes it difficult to restructure the abstract representations during the restructuring stage. Secondly, obtained abstract representations have higher level of detail and complexity (Nugroho, 2009). This means that there are several retrieved elements that might have been omitted to reduce the complexity of abstract representation and, therefore, improve its understandability (Gemino & Wand, 2005; Reijers & Mendling, 2010).

This chapter proposes a business-awareness program comprehension technique following model-driven development principles. The proposal obtains business process models from an existing software system. Business process models represent the sequence of coordinated business activities supported by the system to achieve the common business goals of a company. Business processes, probably, are the models at the highest abstraction level. This technique does not replace to other program comprehension techniques (like those to obtain system design based on a set of class diagrams) but it complements them. This chapter deals with the usage of both business process models and traditional class diagrams to get a better comprehension. The main implication is that a better comprehension during reengineering of open source systems leads to a better enterprise agility.

The proposal is aided by a tool especially developed to support the technique and facilitate its adoption. The supporting tool makes it possible to conduct a case study involving some open source software systems. The case study demonstrates that the main benefits of business-awareness program comprehension are that it hides some non-relevant details thus the understandability of open source software