Automated software license analysis

Timo Tuunanen · Jussi Koskinen · Tommi Kärkkäinen

Received: 18 August 2008 / Accepted: 25 May 2009 / Published online: 6 June 2009
© Springer Science+Business Media, LLC 2009

Abstract  Software license is a legal instrument governing the usage or redistribution of copyright-protected software. License analysis is an elaborate undertaking, especially in case of large software consisting of numerous modules under different licenses. This paper describes an automated approach for supporting software license analysis. The approach is implemented in a reverse engineering tool called ASLA. We provide a detailed description of the architecture and features of the tool. The tool is evaluated on the basis of an analysis of 12 OSS (open source software) packages. The results show that licenses for (on average) 89% of the source code files can be identified by using ASLA and that the efficiency of the automated analysis is (on average) 111 files per second. In a further comparison with two other open source license analyzers—OSLC and FOSSology—ASLA shows a competitive performance. The results validate the general feasibility of the ASLA approach in the context of analyzing non-trivial OSS packages.

Keywords  Software licenses · Software reuse · Open source software · Reverse engineering · Program comprehension

T. Tuunanen (✉) · T. Kärkkäinen
Department of Mathematical Information Technology, University of Jyväskylä, P.O. Box 35 (Agora), 40014 Jyväskylä, Finland
e-mail: timtuun@jyu.fi

T. Kärkkäinen
e-mail: tka@mit.jyu.fi

J. Koskinen
Department of Computer Science and Information Systems, University of Jyväskylä, P.O. Box 35 (Agora), 40014 Jyväskylä, Finland
e-mail: koskinen@jyu.fi
1 Introduction

The costs of software maintenance activities have traditionally been estimated at 50-75% of total software life-cycle costs (Lientz et al. 1978). Moreover, according to some studies (Seacord et al. 2003) the percentage related to maintenance and evolution is increasing, so the importance of this subarea is clearly great. According to Lehman’s first law (Lehman et al. 1998), the software must be continually adapted or it will become progressively less satisfactory in real-world environments.

This is due to continuous changes in user requirements and in aspects of the technical environment. Successful systems inevitably change, due to pressures from changing requirements (Lehman et al. 1998). Software changes need to be planned and implemented in a controlled fashion, in order to avoid deterioration of the software quality due to undesired side-effects resulting from the changes. The typical problems which make changes harder include poor modularity, poor structure, and poor documentation. For example, instances of textually highly delocalized but logically strongly dependent parts of the program can lead to comprehension problems (Letovsky and Soloway 1986). Moreover, comprehension problems in turn make it harder both to fully and partially reuse software in general. Many legacy systems are also very large investments, containing invaluable business logic and knowledge. Because of this fact, there is a need to reuse their components.

Component-based software reuse is one way to reduce the problems of legacy system maintenance. Generally speaking, the reuse of well-defined and tested components can be particularly beneficial. Reverse engineering techniques can be used for retrieving information that is relevant to the maintenance, reuse, and comprehension of large-scale programs. Most reverse engineering tools are based on parsing the source code and providing abstracted views of system components and their interrelations. The provided views support the tool user in making the right choices and decisions concerning potentially reusable components.

Open source software (OSS) development has some typical characteristics, such as licensing, which need to be considered when one is designing reverse engineering support for reuse. In fact, software license analysis is a pre-condition for legally taking components for reuse, or for modification for a specific purpose. The potential effects of having only limited rights to reuse and modify software components need to be taken into account in some manner, and for large systems this is a non-trivial process. This being so, it is helpful to have automated tools that can provide information regarding the use of the licenses and other general views of the software.

Detailed empirical results characterizing the reuse-based software development of large-scale systems have been provided by Selby (2005). That study showed the importance of some module design factors (including few calls to other system modules from a module and many calls from it to utility functions). The module implementation factors that characterize module reuse without revision were small size in source lines and (after normalization by size in source lines): low development effort and many assignment statements. The study also underlined the extent to which a low proportion of faults will minimize the need for revision of modules that are candidates for reuse.

In making decisions on the potential reuse of software components (and modules), it is necessary to take licensing into account (an aspect not investigated by Selby