Retrieving Open Source Software Licenses

Timo Tuunanen[1], Jussi Koskinen[2], and Tommi Kärkkäinen[3]
1,3 Department of Mathematical Information Technology, University of Jyväskylä, P.O. Box 35, 40014 Jyväskylä, Finland,
timtuun@jyu.fi, tka@mit.jyu.fi
2 Department of Computer Science and Information Systems, University of Jyväskylä, P.O. Box 35, 40014 Jyväskylä, Finland, koskinen@cs.jyu.fi

Abstract. Open Source Software maintenance and reuse require identifying and comprehending the applied software licenses. This paper first characterizes software maintenance, and open source software (OSS) reuse which are particularly relevant in this context. The information needs of maintainers and reusers can be supported by reverse engineering tools at different information retrieval levels. The paper presents an automated license retrieval approach called ASLA. User needs, system architecture, tool features, and tool evaluation are presented. The implemented tool features support identifying source file dependencies and licenses in source files, and adding new license templates for identifying licenses. The tool is evaluated against another tool for license information extraction. ASLA requires the source code as available input but is otherwise not limited to OSS. It supports the same programming languages as GCC. License identification coverage is good and the tool is extendable.

1 Introduction

The relative amount of the costs of *software maintenance and evolution* activities has traditionally been 50-75% of the software life-cycle, in case of successful systems with long lifetime [12]. Moreover, according to some studies [21] the relative amount is increasing, so the importance of this subarea can hardly be over-emphasized. According to *Lehman's first law* [11] software must be continually adapted or it will become progressively less satisfactory in "real-world" environments. Many *software systems* have been very large investments, and they contain invaluable business logic and knowledge. Therefore, there is a need to reuse their components.

Component-based software reuse is one way to reduce the problems of software system maintenance. Adaptation of the components, however, can be relatively demanding. For example, the applied *software licenses* need to be taken into account when designing support for reuse. *Reverse engineering* is the main automated general approach for retrieving relevant information for supporting maintenance, reuse and comprehension of large-scale programs. Most of the reverse engineering tools provide abstracted views of system components and their interrelations. This supports the tool user to make right choices and decisions concerning potentially reusable components.

The paper is organized as follows. Section 2 shortly describes the general central problems of software maintenance and nature of reverse engineering approaches. Section 3 describes specific characteristics and problems of OSS maintenance and reuse. Section 4 describes an automated reverse engineering approach and its

Please use the following format when citing this chapter:
Tuunanen, T., Koskinen, J., and Kärkkäinen, T., 2006, in IFIP International Federation for Information Processing, Volume 203, Open Source Systems, eds. Damiani, E., Fitzgerald, B., Scacchi, W., Scotto, M., Succi, G., (Boston: Springer), pp. 35-46

implementation, called *ASLA* (Automated Software License Analyzer), for retrieving relevant license information from source code modules. Short description of the approach has been accepted to the software maintenance community's conference: CSMR 2006 [22]. This paper considerably extends that earlier paper, especially by addressing the issue of license retrieval from OSS perspective, and by providing a more detailed description of ASLA. The tool users are mainly component engineers, software reusers, and software maintainers. The approach and its implementation are not restricted to OSS. However, OSS is a natural setting for developing and testing the approach. OSS is a good source of reusable components, and provides many licenses and their versions. Tool user needs, system architecture, tool features, and tool evaluation are presented. Finally, Section 5 draws the conclusions.

2 Reverse Engineering

Maintaining and reusing large-scale software systems is demanding especially if documentation is inadequate or misleading. While solving maintenance problems, maintainers have information needs [10]. One of the main problems is the identification and comprehension of relevant pieces of programs, and their dependencies. Reverse engineering tools extract that information from the source code and store it into a program database. The extraction is usually achieved by calling a parser component, implemented according to the well-established conventions of compiler construction [1].

Five-level classification of the information retrieval features of reverse engineering tools is provided in [10]. That classification will be later applied in the evaluation part of this paper (Section 4.4). The levels of the model are:

L1. Formation of basic internal data structures (such as abstract syntax trees).
L2. Formation of higher abstraction level access structures (such as call graphs).
L3. Visualization of access structures.
L4. Information request and retrieval mechanisms.
L5. Navigation mechanisms.

Typical features of the main reverse engineering tools are compared in [10]. There are also some other relevant related studies based on structural program analysis and text and documentation analysis, as listed in [9, Appendix 1, Categories 1-3].

3 Characteristics of Open Source Software

Definitions for OSS-related terminology are provided in [19]. OSS community provides a rich base of potentially reusable software. Unlike the more traditional closed source software (CSS), OSS can be freely accessible, used, modified, and redistributed. OSS development has been studied based on a sample of 406 projects [5]. Most used languages were C, C++, Perl, and Java. Despite the large number of

OSS projects, development effort has focused on a few large projects, such as *Linux*, *Mozilla*, and *Apache* [14].

One important aspect in OSS development is the need for greater maintainability. Based on the analysis of almost 6 million LOCs it was concluded [20] that OSS development will produce legacy systems in much the same way as CSS development. It is stated that 20% of the components will produce about 80% of the maintainability problems. Therefore, the problem-prone modules need to be identified. An empirical study of key success factors in software reuse in general based on 24 projects has been conducted [15]. Reusing OSS neither differs much from reusing other kind of software. Therefore, results received from supporting OSS-development should be quite generalizable to CSS also.

One important problem for partial reuse is that there are over 50 different versions of OSS licenses as listed by Open Source Initiative [19]. GPL is the most common license [5]. License information concerning the dependency of different modules provides the key metainformation for partial reuse. Component-based white-box reuse of OSS is natural, e.g., since license information is typically bound to modules. It is clear that good tool support reduces the reuse and comprehension problems. Reuse can be supported by identifying reusable component candidates, simplifying the license identification, and providing abstracted views of the relevant components and their interrelations.

4 The License Retrieval Approach

There is a clear need for software reuse oriented license analysis. It can be made more effective by automated license identification of source code files by using text searching techniques and by providing information about file dependencies. In this section we present *ASLA*, which is our license retrieval approach for this purpose.

4.1 User Needs

OSS reuse can be classified into two different approaches: Using the whole software package as-is and modifying it and using part of the software packages as part of another program. Both cases introduce three main user needs as presented below.

4.1.1 Identifying Dependencies

There is a huge amount of code for different platforms and not all source code is used in certain platform in large OSS packages such as *Linux* kernel [13]. Therefore, user needs to know what source files are used in a particular environment. When build process outputs are identified the information can be used for component identification. This can give some clues about reusable components inside a larger software package and becomes useful when considering partial reuse. Licenses

behave differently depending on what part of software is dependent on other parts of the software. Therefore, the user must also know what libraries are linked to the program and recognize the dependencies between all objects in order to make reliable license analysis.

4.1.2 Identifying Licenses in Each Source File

OSS is distributed under one or more licenses. Unfortunately all OSS licenses are not compatible with each other and they pose different restrictions so that each source code file must be checked separately. It is vital, at least from the commercial perspective, to check that licenses of a software package are in order to avoid any legal consequences.

4.1.3 Adding New License Templates

In most cases programmers who write OSS use the predefined templates [18] to indicate the use of certain license. Unfortunately this is not the case in all software packages. In many cases license of the source code is indicated in a way that is not known in advance. Therefore, there is an obvious need to add new search criteria for licenses as part of the license analysis.

4.2 System Architecture

Fig. 1 shows the system architecture in UML-notation. *ASLA* employes three open source programs: *GCC* [7] [8], and modified versions of *ld* (linker) and *ar* (archive builder) that are based on *GNU binutils* [4] (version 2.15.97). Any version of *GCC* compiler which supports environment variable `$DEPENDENCY_OUTPUT` can be used.

ASLA is implemented in *Linux* operating system using Java programming language (version 1.5.0_01). *GCC* supports compilation of many programming languages, which are supported by *ASLA* also. Only requirement is that dependency information files (DIFs) produced by *GCC* are available. *Ar* and *ld* are modified in a way that these programs write similar DIFs about dependencies of the libraries as *GCC* does for the source code files and compiled objects. DIFs form the program database. It contains the information about compiled and linked objects and their dependencies. DIFs serve as a basis for data integration between these four programs. *ASLA* reads the DIFs, analyzes licenses of files listed in them, creates a dependency map based on them and visualizes the information.

Fig. 2 presents the contents of *ASLA* user interface after analysing *gaim* [6] (version 1.2.1), which is an open source instant message client. It is used as the main example case in this paper. The left panel of the figure shows hierarchically the analyzed file structure. The modules can be selected from it and opened to the right panel for viewing their contents.

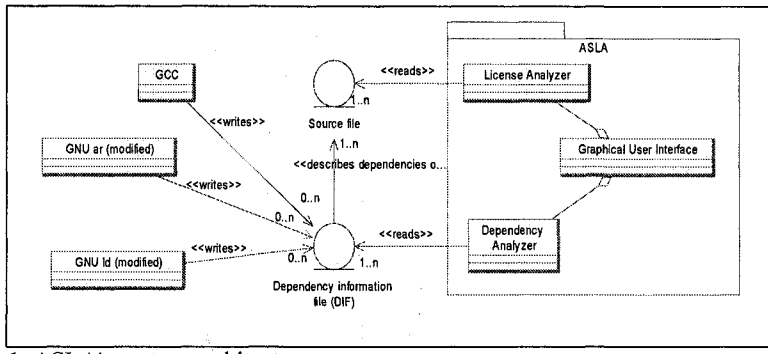

Fig. 1. ASLA's system architecture

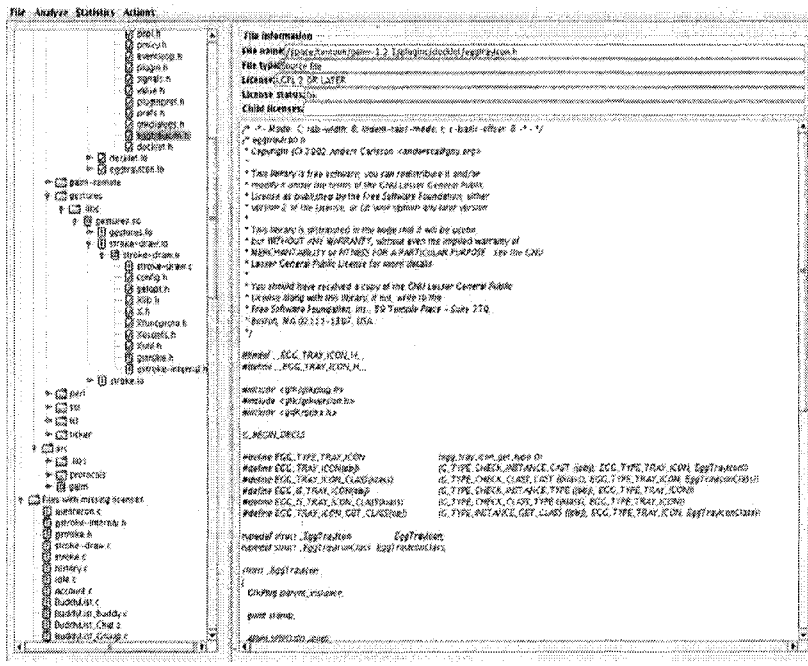

Fig. 2. ASLA after dependency and license analysis of *gaim*

4.3 Features

The implemented features of *ASLA* as described below are based on the user needs introduced earlier.

4.3.1 Identifying Dependencies

ASLA produces dependency map of source code, build process outputs and linked libraries by using the DIFs. Dependency map is a data structure where all objects described in DIFs are stored. Each file has references to the objects that the file is dependent on and to the objects that are dependent on it.

Each object file that is compiled is, at least, dependent on the initial source code file(s) and all source code files that are included or referenced from them. This information is given in *GCC* dependency output as follows:

```
.libs/irc.o: irc.c ../../../src/internal.h
../../../config.h ...
```

This output tells that compiled object `irc.o` (in directory `.libs`) is dependent on (i.e. includes) source files: `irc.c, internal.h and config.h` etc.

To identify what compiler outputs (and source code files) are included in the software, there must be information about what compiler outputs are linked to each executable or library. This information can not be reverse engineered from binary files (linker outputs) so it is collected during the build process using *ld* and *ar*. The following dependency output is obtained from *ld*:

```
.libs/libirc.so: /usr/lib/crti.o .libs/irc.o
```

This output tells that shared object `libirc.so` (in directory `.libs`) is dependent on (i.e. includes) object files: `crti.o and irc.o`.

For each DIF the following operations are performed by *ASLA*:

- Reading the file name of the target object (for example `libirc.so`).
- Adding the target object to the dependency map if it does not exist.
- Reading all child objects' file names.

For each child object:

- Adding the child object to the dependency map if it does not exist.
- Setting the target (`libirc.so`) object as a parent object; each object can have multiple parents.
- Adding the child object as parent's child.

This algorithm produces the dependency map described above. Each compiled object gets it's license as collection of it's children's licenses. If license changes are made to objects from hereon they are visible to all parent and child objects instantly.

4.3.2 Identifying Licenses in Each Source File

ASLA automatically identifies licenses of single source code files. This is achieved by using license templates that are compiled into regular expressions (in BNF) as described below.

Most simple open source licenses, such as BSD or MIT are usually written in the beginning of the source code file. Another way to indicate the license of the source code is to reference the license from the source code. This technique is used for example in GPL and LGPL licenses [18].

In the source code file one can either find a simple notification such as: `For license information: see file COPYING`, or a defined template text that

indicates the license of the source file. `COPYING` is the typical name of the license file in OSS.

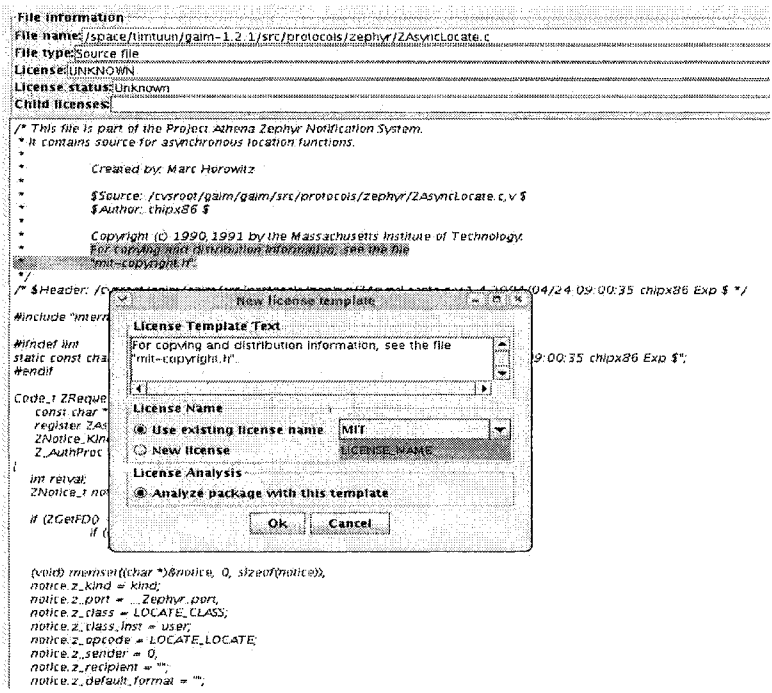

Fig. 3. ASLA after adding a new license template for *gaim*

Identifying licenses of source files that contain pre-defined template or full license text is fairly simple relying on finding the predefined text from source code file. This approach, however, requires that all unnecessary source code characters (such as comment characters) are removed and different white space characters are allowed between words.

Many programmers modify the predefined texts slightly and there are also many different versions of licenses published. For example LGPL was previously called *GNU Library general public license* and nowadays it is called *GNU Lesser general public license.* Therefore, there are many slightly different texts within source code indicating the same license. Hence, their recognition requires more sophisticated text searching techniques. Especially, regular expressions can be used for allowing white space characters, alternative words and undefined characters.

For example, *ASLA*'s license search template for LGPL (version 2 and 2.1) is the following:

```
... GNU (Library)|(Lesser) General Public License as
published by the Free Software Foundation; either version 2.*,
...
```

This is compiled into a regular expression:

```
...\s*GNU\s*(Library)|(Lesser)\s*General\s*Public\s*License\s
*as\s*\s*published\s*by\s*the\s*Free\s*Software\s*Foundation;\s
*either\s*version\s*2.*,\s etc.
```

This is interpreted as follows: "0..n white spaces", "GNU", "0..n white spaces", "Library or Lesser", ... , "version", "0..n white spaces", "2", "0..n any character", ",", ...

Unfortunately license of every single source code file can not be reliably identified and, therefore, user must have a possibility to identify licenses also manually. Such feature is supported by *ASLA*. First way to do manual identification is to apply license of the separated license file for all source files in subdirectories of the directory where the file is found. This technique is useful in a situation where license file is meant to cover all files in subdirectories but source files themselves do not include any reference to the used license.

Another way to do manual license identification is to manually check all unidentified source files. This is aided by *ASLA* that lists all source code files that were unidentified separately. To reduce the number of unidentified licenses and need for manual license identification with other software packages the tool user is able to add new license templates.

4.3.3 Adding New License Templates

ASLA offers two different ways to introduce new license identification templates. First way is to create new text file into the directory where existing license template files are saved. File format for new template contains the license name on the first line of the file and template text in regular expression form on the following lines. Another way is especially usable. User is able to select a text in a source file and use that text as a license identification template (Fig. 3). In this case *ASLA* forms the regular expression automatically.

4.4 Evaluation

In this section *ASLA* is evaluated against *LIDESC* [17], which is another license information extractor. *ASLA* and *LIDESC* have many similarities but the focus areas and applied techniques have their differences. *ASLA* is targeted especially for component engineers, and other reuse and maintenance personnel. The approach is extendable and designed to be used for analyzing existing software packages. An especially rich base of possibly reusable software is OSS packages. *ASLA* itself has also been implemented based on reusable OSS components.

As an example of used source code we consider *gaim* which includes total of 506 source files. 437 (86%) of them were used in the selected test environment (*Linux*). *ASLA* does not require any makefile modications to produce DIFs. Existing software

packages can be analyzed as they are. In *LIDESC* all source files must be compiled using defined compiler flags. The user must manually modify all makefiles or define the parameters in *autoconf* [3] scripts. From user perspective this is probably not the preferred approach, especially, when analyzing large potentially reusable software packages.

4.4.1 Identifying Dependencies

DIFs contain information of dependencies, which is the basis for forming basic level data structures. This corresponds to level L1 of tool features as presented in section 2. Both *ASLA* and *LIDESC* naturally form internal data structures.

Information contained in the DIFS in *ASLA* is collected and combined in order to create higher abstraction level access structures (level L2). This is done by the *ASLA* dependency analyzer when creating the dependency map based on the DIFs. Features of this level are not convincingly reported for *LIDESC*.

The dependency map is visualized by *ASLA* in tree form (level L3). *LIDESC* does not support this level. The information visualized in *ASLA* is useful both in full and partial reuse of software packages. For partial reuse, compiled objects that have no parent objects are potential reusable components. For example, in Fig. 4 all files with extension `.so` (shared library objects) are such compiled objects.

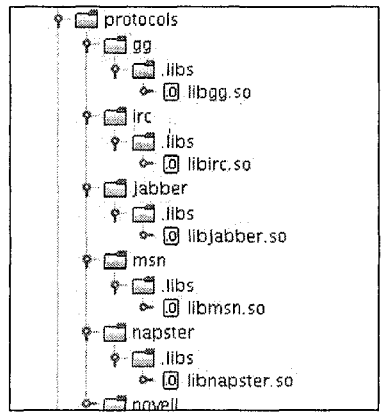

Fig. 4. ASLA's tree for showing the potentially reusable components (*gaim*)

In case of full reuse, the *ASLA* tree format introduces the dependencies of the different parts of the software and indicates how licenses of the compiler outputs are collected from the source files. Neither *ASLA* nor *LIDESC* provide real navigation capabilities (level L5), which could be useful in case of complex dependencies, although *ASLA*'s file tree can be browsed and direct access to the files is provided.

4.4.2 Identifying Licenses in Each Source File

Information requests (level L4) are supported in *ASLA* based on regular expressions. Therefore, the approach adapts well into "real world" of varying OSS packages. *LIDESC* is implemented in a similar way but is based in this regard on exact match of license identification string in the source file. Due to alternative word matching, and ability to handle undefined characters and different commenting styles, *ASLA* provides more flexibility. It handles the identification of modified and different versions of licenses without need to introduce new identification templates for each different license version.

The license identification coverage of *ASLA* against *LIDESC* can be further compared with our *gaim* example case. On our initial analysis we were able to identify license of 315 source files out of 437 (72%) using 7 different license search templates. The reason of the moderate identification ratio was that one *gaim* component did not contain any references to used licenses in source code. To reach the same result using the exact matching technique of *LIDESC* would have required at least 20 unambiguos license identification strings.

Manual license identification, which is not supported by *LIDESC*, complements the license analysis in our example case of *gaim*. By applying the license found in the file `COPYING`, which was explained earlier and which can be found on top directory of the component, to the files of the component, we were able to identify licenses of 350 files out of 437 (80%).

Moreover, *ASLA*'s initial analysis of *Mozilla* [16] identified licenses of 5654 files out of 5871 (96%) using 10 different license templates and licenses of 283 files out of 301 (94%) of *Apache http server* [2] using 5 different templates. These results illustrate both good coverage and scalability of *ASLA*.

4.4.3 Adding New License Templates

Final step in our *gaim* example was for the user to introduce a new license template during the license retrieval (as presented earlier). In our case it was the following: `For copying and distribution information, see the file "mit-copyright.h"`. When this template was introduced and used in the analysis the final number of identified source files was 401 out of 437 (92%). By comparison *LIDESC* does not support addition of new license templates during the license retrieval. Another way of new license template addition is to add new file entry to license template directory. This offers more versatile but more complex way since the template must be in BNF. *LIDESC* applies a similar approach. However, in that case the license must be in a specifically formatted text file and it must be added using specific seven step process as described in *LIDESC* documentation.

5 Conclusions

This paper has presented a license retrieval approach and its implementation called *ASLA*. It is targeted at retrieving software license information from source code modules. At general level it has been motivated by the characteristics, problems and needs of OSS development, maintenance and component reuse. License retrieval and comprehension is especially important for effective component reuse. It can be concluded that *ASLA* addresses an important problem. *ASLA* has been tested and compared to *LIDESC*, which is another known license information extractor. *ASLA* provides promising results regarding the coverage of identified licenses, and supported information retrieval levels as compared to *LIDESC*. *ASLA* uses regular expressions and dependency information files (DIFs). The approach was found sufficiently effective, and can be applied to several programming languages. Incorporation of new licenses is uncomplicated by using the license templates. The applicability of the approach is neither restricted to OSS. Further research avenues include studies regarding information abstraction and visualization, e.g. architectural views, handling of the even more complex cases of license determination in case of multiple applied licenses, and system efficiency optimizations.

References

1. Aho, A.V., Sethi, R., Ullman, J.: *Compilers - Principles, Techniques, and Tools*. Addison-Wesley (1986)
2. APACHE HTTP Server Project. http://httpd.apache.org/ (accessed 25.9.2005)
3. Autoconf. http://www.gnu.org/ software/autoconf/ (accessed 13.9.2005)
4. GNU Binutils. http://www.gnu.org/ software/binutils/ (accessed 13.9.2005)
5. Capiluppi, A., Lago, P., Morisio, M.: Characteristics of Open Source Software Projects. Proc. 7th European Conference on Software Maintenance and Reengineering (CSMR 2003) 317-330. IEEE Computer Soc.
6. Gaim: A Multi-Protocol Instant Messaging (IM) Client. http://gaim.sourceforge.net/ (accessed 19.9.2005)
7. GCC: GNU Compiler Collection. http://gcc.gnu.org (accessed 13.9.2005)
8. GCC 4.0.1 Manual. http://gcc.gnu.org/onlinedocs/gcc-4.0.1/gcc/. Free Software Foundation (2005) (accessed 13.9.2005)
9. Koskinen, J.: Automated Transient Hypertext Support for Software Maintenance. Jyväskylä Studies in Computing 4 (2000). Univ. of Jyväskylä, Jyväskylä, Finland
10. Koskinen, J., Salminen, A., Paakki, J.: Hypertext Support for Information Needs of Software Maintainers. Journal of Software Maintenance and Evolution: Res. and Pract. 16, 3 (2004) 187-215
11. Lehman, M., Perry, D., Ramil, J.: Implications of Evolution Metrics on Software Maintenance. Proceedings of the International Conference on Software Maintenance - 1998 (ICSM 1998) 208-217. IEEE Computer Soc.
12. Lientz, B., Swanson, E.: Problems in Application Software Maintenance. Communications of the ACM 24, 11 (1981) 763-769
13. The Linux Kernel Archives. http://www.kernel.org (accessed 13.9.2005)

14. Mockus, A., Fielding, R., Herbsleb, J.: Two Case Studies of Open Source Software Development: Apache and Mozilla. ACM Transactions on Software Engineering and Methodology 11, 3 (2002) 309-346
15. Morisio, M.: Success and Failure in Software Reuse. IEEE Transactions on Software Engineering 28, 4 (2002) 340-357
16. Mozilla. http://www.mozilla.org/ (accessed 19.9.2005)
17. Nordquist, P., Petersen, A., Todorova, A.: License Tracing in Free, Open, and Proprietary Software. Journal of Computing Sciences in Colleges 19, 2 (2003) 101-112
18. Opensource Org.: The Approved Licenses. http://www.opensource.org/ licenses/ (accessed 16.9.2005)
19. Perens, B.: The Open Source Definition. http://www.opensource.org/ docs/definition.php. Open Source Initiative (2005) (accessed 13.9.2005)
20. Samoladas, I., Stamelos, I., Angelis, L., Oikonomou, A.: Open Source Software Development Should Strive for Even Greater Code Maintainability. Communications of the ACM 47, 10 (2004) 83-87
21. Seacord, R., Plakosh, D., Lewis, G.: Modernizing Legacy Systems: Software Technologies, Engineering Processes, and Business Practices (2003). Addison-Wesley
22. Tuunanen, T., Koskinen, J., Kärkkäinen, T.: ASLA: Reverse Engineering Approach for Software License Information Retrieval. Accepted to CSMR 2006.