

License Compliance Software as a Tool for Open Source Risk Management

Ville Oksanen, Helsinki University of Technology and Mikko Välimäki, Hanken Swedish School of Economics, Finland

Introduction

Several studies have confirmed that real and perceived legal risks are one of the major obstacles for large-scale adoption of open source products in the corporate environment. Especially intellectual property (mainly copyrights and patents) risk management has become important as the corporate use of open source is increasing and open source start-ups have become targets for mergers and acquisitions. One of the main sources for intellectual property risks are compliance problems with open source licenses: large open source software packages may include components whose license terms are incompatible with the rest of the package.

This article describes how license management can tackle the incompliance problem. First, we discuss the increasing relevance of license violation risks in open source development. Second, we explain briefly how different companies and communities are in practise trying to keep control of these risks. Then, we document an experiment with developing an automated license checking and analysing software. Our research project has developed a tool, which retrieves license information inside source code software packages and lists all identified licenses. We describe the design of the software and the outcomes that can be achieved by using it. We also explain the limitations of the current version and compare it to commercial alternatives. Finally, we discuss how the software could be improved to be more useful as part of intellectual property risk management strategy of software companies.

In the end, we argue that technical solutions can indeed help minimizing the risks related to unintentional open source license conflicts. In the long term, however, intellectual property risks based on incompatible licenses could be minimized more effectively by reducing the number of open source licenses and by changing the existing licenses to be more compatible with each other. It must be acknowledged, though, that internal license compatibility between open source licenses is only part of the problem: there remains always the risk that a large package may include components under proprietary licenses, or that some part of the package may infringe other intellectual property rights such as the patent law.

License Management and the Incompliance Problems

Rising Risk Factor

Ever since the widely reported legal actions by SCO against major Linux supporters there has been a growing need for intellectual property risk management among open source users and developers. Typical infringement worries include the following: (1) third party source code has been illegally copied into an open source project, (2) license terms of third party components are not followed and (3) third party patent rights are being infringed. Further, the knowledge of the law is not enough and also concrete acts are called for to protect communities and also increasingly companies against the possible consequences of intellectual property violations.

The current legislative regime has become rather strict towards any kind of intellectual property violation. This is mainly because major music and movie production companies and their lobby organizations have succeeded in setting the agenda for the international legal development. From their perspective the main goal has been to fight “piracy” – to go against anonymous offshore CD-printers and individual file-sharers on the Internet – which obviously requires quite a different toolset than dispute resolution in traditional business-to-business relationships.¹

As a consequence of this regulatory development any intellectual property violation in commercial scale is a cause for criminal prosecution both in Europe and the United States. For example, in Finland a commercial copyright violation is carrying a maximum punishment of two years in prison and in the United States it is up to ten years. In addition to the possible imprisonment, the sanctions also include large fines, other monetary compensation and the destruction of infringing products as well as manufacturing devices including computers.

It should be noted that even non-commercial copyright violations are increasingly carrying severe criminal sanctions. For example, the criminal law was changed in Finland in the beginning of this year to make any copyright violation, which takes place in “a network” and has a potential to cause significant damage to the right holder, to carry the maximum penalty of two years in prison. In the United States the 'No Electronic Theft' Act was passed already in 1997 and it carries penalties of up to five years in prison and up to \$250,000 in fines. (Bartow, 2004).

Even the non-criminal sanctions against intellectual property violations can be devastating for both communities and companies alike. If a court orders a permanent injunction against a central component of a project based on a copyright or patent violation, minor participants may not have the required resources to correct the situation. Even the cost of hiring a lawyer is often too much for community lead projects and small enterprises. Thus, avoiding risks and violations in the first place should be the soundest strategy. However, this is easier said than done. As illustrated in the Figure 1, the internal logic of intellectual property laws mean that

¹ In the software industry, organizations such as Business Software Alliance have been active in this debate.

developers and users can be liable for even those violations, which have taken place during the upstream development.²

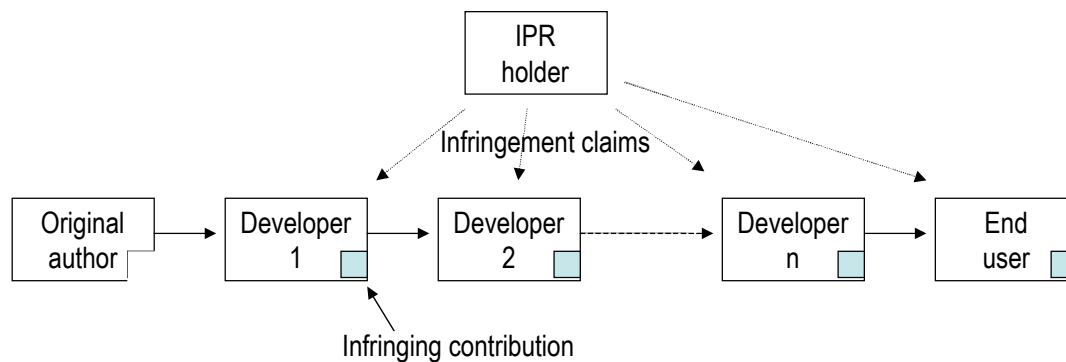


Figure 1. Developer-chain and intellectual property infringement (Välimäki & Oksanen 2005)

Risk Management in Open Source Environment

To counter the rising risk level, open source projects (and companies) have increased the use of different risk management strategies. Some of the better-managed projects and companies established such strategies already in the 1990s. For many developers, SCO's highly public cases against IBM and other Linux supporters launched in 2003 was the final wake-up call that open source software is not in any legal safe-haven. Indeed, the nature of open source means that intellectual property risk management can be more complex than before. Distributed development causes additional uncertainties compared to the traditional model, in which it is possible to keep the developers under stricter control. (c.f. Välimäki & Oksanen 2005 for more detailed analysis of the topic)

There are several possible approaches to risk management. In the end, financial constraints are likely to dictate the choices. For example, launching a major lobbying campaign to limit patent liability is typically out of the reach of smaller companies and communities but may well be a good investment for companies like Intel and Cisco, which indeed are hoping to change the patent litigation regulations in the USA. (Lombardi 2006) Similarly, intellectual property insurances and the establishment of a defensive patent portfolio are options, which are too expensive and too complex to manage for most small-scale developers.

The most popular and also generally speaking the cheapest risk management tools are related to risk avoidance. For example, staying out of areas of software development, which are heavily patented (e.g. colour management) or otherwise legally in the grey area (e.g. peer-to-peer applications) lowers significantly the risk being sued by someone. Similarly, a policy to not accept code from unknown developers reduces the risk of infringing code. A cynical

² To be precise, the consequences of accidental infringement are in practise much more limited and can in some cases be totally avoided. However, in principle it does not matter whether the infringer for example knew about an infringing patent.

observer would also note that staying poor belongs to this category since lawsuits are rarely raised without a profit-motive. (Graham 2006).

In addition, most of the open source licenses use standard liability disclaimers in their licenses. While disclaimers may be somewhat effective in limiting liabilities pertaining to damages caused by a malfunctioning software, they offer little to none protection against third party intellectual property violation claims. Correspondingly, disclaimers may not protect companies in all jurisdictions if their customers are consumers.

The summary of the risk management options can be found from table 1.

Table 1. Different intellectual property defence options for open source developers (Välimäki & Oksanen 2005)

	Scope	Effectivity	Speed	Price
Disclaimers	Licensees	Low	Fast	Low
Insurance	Market	Medium	Fast	Medium to High
Patenting	Market	High	Slow	High
Avoidance	Market	Medium	Fast	Varies
Lobbying	Regulatory	Medium to High	Slow	Medium to High

License Incompatibility Problem

The open source “economy” has also its own peculiar problems, which are to a large extent caused by the current high number of incompatible open source licenses. Open Source Initiative (OSI), which grants “OSI Certified” marks for open source licenses, lists currently almost sixty separate licenses. The problem is that most of the reciprocal licenses are incompatible with each other: open source code licensed under one approved reciprocal license may not be used in a project licensed under another approved reciprocal open source license (e.g. Rosen 2004).

There are no good reasons for this many incompatible licenses as many of them are just slightly differently worded copies of each other. Therefore it is very unfortunate that OSI’s project to curb the license proliferation has been so far a failure. Only five licenses have been de-recommended by this date (i.e. Sun Industry Standards Source License, Intel Open Source License, the Jabber Open Source License and old versions of Academic Free License and Open Software License; see Majerus 2005)

The license proliferation makes it very difficult to keep track on which licenses are compatible with each other. Even lawyers specialised in open source may not be able to answer right away to more exotic queries like “is it legal to combine two files, which are licensed with Nethack General Public License and Motosoto License, respectively”. Still, most of the projects tend to use a limited set of licenses, which helps keeping the cost of license management at reasonable level.

Another set of license related problems is caused by the licenses administered by the Free Software Foundation (FSF). The fuzzy reciprocal nature of General Public License (GPL) has made it difficult to know exactly to what extent and under what circumstances it is possible to combine GPL'ed code with other code, which is not licensed with a Free Software License as defined by FSF. (This topic has been already extensively studied, see e.g. Välimäki 2005, 124-130; Rosen 2004, 103-141). The problem with GPL has been highlighted e.g. in the recent cases dealing with the proprietary drivers for the Linux Kernel (Zonk 2006, Smart 2006, Novell 2006).

Compatibility issues related to Lesser General Public License (LGPL, FSF's second most popular license) are typically not as well known as the problems related to GPL. Many developers tend to think that LGPL allows always linking and only direct changes to LGPL'd file require sharing the outcome with the public. However, this is not necessarily the case due to the complex requirements set in Article 6 of LGPL:

6. As an exception to the Sections above, you may also combine or link a "work that uses the Library" with the Library to produce a work containing portions of the Library, and distribute that work under terms of your choice, provided that the terms permit modification of the work for the customer's own use and reverse engineering for debugging such modifications.

.....

Also, you must do one of these things:

** a) Accompany the work with the complete corresponding machine-readable source code for the Library including whatever changes were used in the work (which must be distributed under Sections 1 and 2 above); and, if the work is an executable linked with the Library, with the complete machine-readable "work that uses the Library", as object code and/or source code, so that the user can modify the Library and then relink to produce a modified executable containing the modified Library. (It is understood that the user who changes the contents of definitions files in the Library will not necessarily be able to recompile the application to use the modified definitions.)*

** b) Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (1) uses at run time a copy of the library already present on the user's computer system, rather than copying library functions into the executable, and (2) will operate properly with a modified version of the library, if the user installs one, as long as the modified version is interface-compatible with the version that the work was made with.*

** c) Accompany the work with a written offer, valid for at least three years, to give the same user the materials specified in Subsection 6a, above, for a charge no more than the cost of performing this distribution.*

** d) If distribution of the work is made by offering access to copy from a designated place, offer equivalent access to copy the above specified materials from the same place.*

** e) Verify that the user has already received a copy of these materials or that you have already sent this user a copy.*

That is difficult to interpret by developers, or as Rosen (2004, 123-124) puts it:

"These sections of the LGPL are an impenetrable maze of technological babble. They should not be in a general purpose software license"

License Compliance Software

Background

Our research project at the Helsinki University of Technology set a goal to create a tool, which would automate some parts of intellectual property risk management. At the first stage, we chose to concentrate on license compliance issues. Automated license analysis makes it considerable faster and easier to find out what licenses and possible problems exist in open source software packages.

A typical open source software package has hundreds or thousands of files, whose copyright holders may have freely chosen the license(s). This means that it is not enough to know what the “main license” of a project is. It is a common misperception that if software is released e.g. as “GPL”, all the files will fall automatically under GPL and there is no need to check the files on individual level. In reality, most of the packages carry files with several licenses. Sometimes these licenses are incompatible with each other.

A manual license check for thousands of files is tedious and mind-numbing task. Still, for example in a merger or acquisition situation there may be explicit regulation that requires one to find out such legal liabilities. Somewhat surprisingly, we were not able to identify any relevant open source tool, which would help in this process. Some commercial closed-source tools are available, which offer more extensive solutions e.g. advanced file recognition and source code comparison feature to detect also code copying.³ However, in addition to being pricey, their working logic is not disclosed to the extent one could be fully sure how the check is made and whether there are any gaps in their analysis.

Thus, a group of students who participated into our institute’s software project course wrote the first version of our “Open Source License Checker”.⁴ The group was supposed to use 150 hours per student on the project, which was estimated to be enough to create a fully working demo version. Unfortunately, the group did not have enough persons specializing in coding and the outcome did not fulfil all the goals we set in the beginning of the course. That said, we believe that the outcome is still something that really helps in the risk assessment and license compliance analysis.

Technical Description of the License Compliance Software

Functionality

First, the program detects licenses in individual source code files and then compares them to its license database. Most of the source files include rather standardized copyright and license information in the beginning of the file. However, license detecting is not enough for detailed analysis. The software should be able to understand also the links between the files because

³ See the products of e.g. Black Duck Software <http://www.blackducksoftware.com/> and Palamida <http://www.palamida.com/>. The usefulness of code comparison is limited to the extent that it can be only done to code already available – in practise other open source code.

⁴ Available at <http://sourceforge.net/projects/oslc>

two files under incompatible open source licenses should not have links between them. The linking information is also readily available since it has to exist for the compiler. The harder task is to describe the incompatibilities and also take into consideration the special rules about linking regarding each license.⁵ The current version of the tool does not yet recognize the links, but it has already partial incompatibility information about the licenses stored as part of the database.

The tool also detects files, which totally lack license information and files that only refer to e.g. License.txt. It also detects files, which have non-standard licenses or slightly modified versions of the OSI-certified open source licenses. As a consequence, the tool also verifies the licenses for any changes and additions (negative check, not currently activated).

After the licenses are recognized and analysed, the tool shows the collected information in easily comprehensible format. The tool allows exportation of the information in such a format that the tool can be added to automatic processes, which monitor e.g. the source code management of the project.

Internal Structure

Information about the license types is stored in a license database, which is implemented as an XML file and separate text files for the actual license texts. When used, the XML file is read into memory, from where it can be efficiently accessed. The license information is stored in objects. A functionality for automatic database updates is currently not available but it could be added to the software at a later stage.⁶ The XML-file is easily extended and the rules about compatibility may be also updated by hand.

The software recursively parses through the specified directory structure, and tries to match each source code file with some license from the license database. This is done by trying to find the keywords associated with a particular license from within the source file. If the source file does not match to the license, the process is repeated until every license type in the database has been tried. A possibility for more advanced recognition heuristics exists in the future versions. – The possible license match for each file is then stored to generate a clear looking tree structure of the results in the GUI.

The license database XML-file uses the acronym of the license as a database key. The XML-file is used for license information storage, but the actual license texts are stored as separate text files. The XML-file contains the file names of the text files. This allows the system to show the actual license text to the user when needed.

The XML file has the following element structure:

```
* licensedb
  o license
```

⁵ For example, GPL allows linking to proprietary libraries, which are a standard part of the operation system.

⁶ For example, a company could have internal rules about allowed mixing of the licenses, which could be changed on the fly.

- + name
- + shortname
- + licensefile
- + shortlicensefile
- + compatibility
 - # class1
 - # class2
 - # class3
- + keywords
 - # group
 - * keyword
- + haropallo

The main XML element in the license file is the 'licensedb'-element. It has 'license' child elements which in turn have the child elements described by the above list. The exact contents of the elements are described in the Appendix 4.

User Interface

The aim has been to develop an easy-to-use tool so that even non-technical persons such as many legal professionals can effectively use it. Thus, the tool has a wizard-like user interface, which has four separate windows. In the first window the user is able to choose the target for the inspection. (Appendix 1.) This can be either directory or a compressed file. At this stage, the software does not support any Linux package formats, but this can be easily implemented. After the file is selected, the second window shows a graphical indicator for the license scanning process. (Appendix 2.)

The results are presented in the third window. (Appendix 3.). From there, the user gets general information about licenses in the scanned object (e.g. what licenses there are and are there files, which do not have any licenses). For example, in Appendix 3. the package has 13 recognized source files, which are all licensed with the Apache-license.

The software shows the files in interactive tree-structured format with individual license information. The user can see the contents of individual files by clicking the file in order to manually check its contents. The user can also print or save the result and check the full license text (the fourth window). The design makes it easy to find the problematic files from the package. However, it could be still improved by adding a possibility for filtering the files based on the license.

The software works also from the command line. It has only one parameter i.e. the name of the inspected target. This makes it possible to use the software for automated scripts.

Practical Experiences

Even though the entire design goal was not met, the tool works already well enough for real-world experimenting. The outcomes differ considerably between well-managed packages and packages, which are done by less meticulous communities. For example, it is quite easy to

find license violations in popular open source peer-to-peer software packages. To compare, none of the checked packages distributed by the Apache-foundation had any questionable licensing combinations.

The single major problem detected seems to be the lack of licenses. There are lot of packages, which only carry on one license in the readme.txt or license.txt. If a file is taken from this kind package to another package, that may easily cause involuntary change of the license. Thus, in the case of open source, all files should always carry a license notification.

Conclusions

Intellectual property infringement risks have to be taken seriously in the development and use of open source software. While the general legal knowledge about the problems is increasing, there is still lot of room for improvements to manage the risks in practice.

The license compliance software we have developed can be used as a tool to manage license-related risks. It helps both community projects and company users to analyse licenses used in open source packages and improve general understanding of open source licensing. Still, the tool is far from a one-size-fits-all solution to manage intellectual property infringement risks: for example, the software can not be used to detect code copying or patent infringements. Further, the functionality of the compliance tool could be improved. The next logical step would be to implement internal linking analysis for incompatible open source licenses.

In our opinion, licensing-related legal risks could be also minimized in the long-term through stopping the proliferation of incompatible licenses. Perhaps OSI or some other recognized “legal standardization” could also create a guide for the marking of the license information in the source files. This could include e.g. a list for hashed versions of the licenses for efficiency (both space and recognition). Unnecessary license incompatibility situations could be also avoided if reciprocal licenses would recognize each other (currently for example Eclipse public license does this⁷).

Finally, we feel that our tool could be used in future empirical research about license compliance in open source development. For example, it would be interesting to analyse which kind of projects tend to have compliance problems and how these problems are resolved when detected.

References

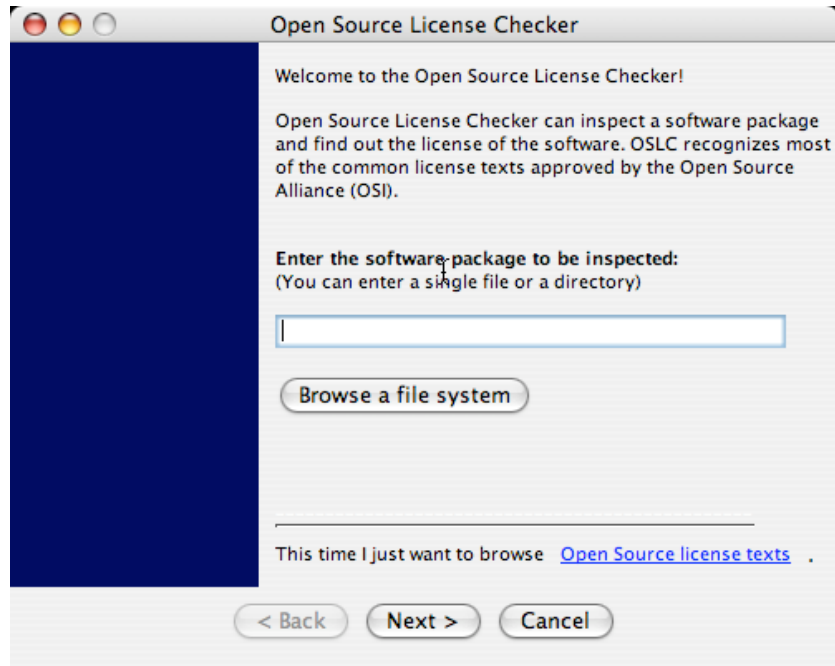
Brown, S.L., Tilton, A. & Woodside, D.M. The case for online communities. The McKinsey Quarterly, <http://www.mckinseyquarterly.com/article_page.asp?ar=1143&L2=24&L3=45>,

⁷ According to the license text: “When the Program is made available in source code form: a) it must be made available under this Agreement or any other OSI-certified copyleft license; and b) a copy of the Agreement must be included with each copy of the Program.”

26.5.2002

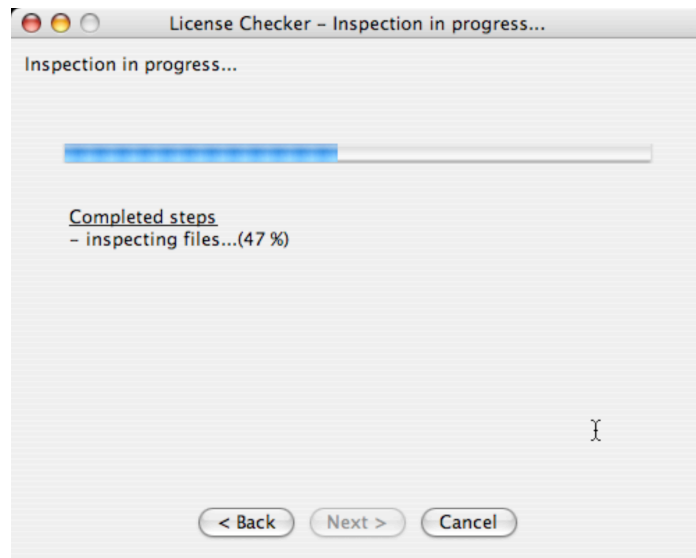
- Bartow, A. 2004. The Hegemony of the Copyright Treatise, 73 U. Cin. L. Rev.
- Graham, P. 2006. Are Software Patents Evil. <http://www.paulgraham.com/softwarepatents.html>
- HP. 2006. FAQ for the Partner Linux Driver Process.
<http://developer.novell.com/wiki/index.php/FAQ_for_the_Partner_Linux_Driver_Process>
- Lombardi, C. 2006. Tech firms to lobby for patent litigation reform. CNET News.com. Published: May 11, 2006, < http://news.com.com/2102-1014_3-6071321.html?tag=st.util.print>
- Majerus, L. 2005. Voluntary De-Recommendation of Sun's SSSL. Email to License Proliferation Committee List. < <http://crynwr.com/cgi-bin/ezmlm-cgi?9:mss:26:200509:kjgjmdbaglfciipdkn>>
- Rosen L. E. 2004. Open Source Licensing: Software Freedom and Intellectual Property Law. Prentice Hall.
- Smart, C. 2006. Kororaa Accused of Violating GPL. Post to website. Saturday, May 13
<<http://kororaa.org/index.php?entry=entry060512-160752>>
- Välimäki, M. 2005. The Rise of Open Source Licensing. Turre Publishing. <<http://pub.turre.com/>>
- Välimäki, M. Oksanen, V. 2005. Minimizing IPR Infringement Risks in Open Source Projects. Software Development - Proceedings of the International Conference on Software Development, May 27 - June 1, 2005, University of Iceland. University of Iceland Press.
- Zonk. 2006. Linux: Kororaa Accused of Violating GPL. News-article.
<<http://linux.slashdot.org/article.pl?sid=06/05/14/2059242>>

Appendix 1.



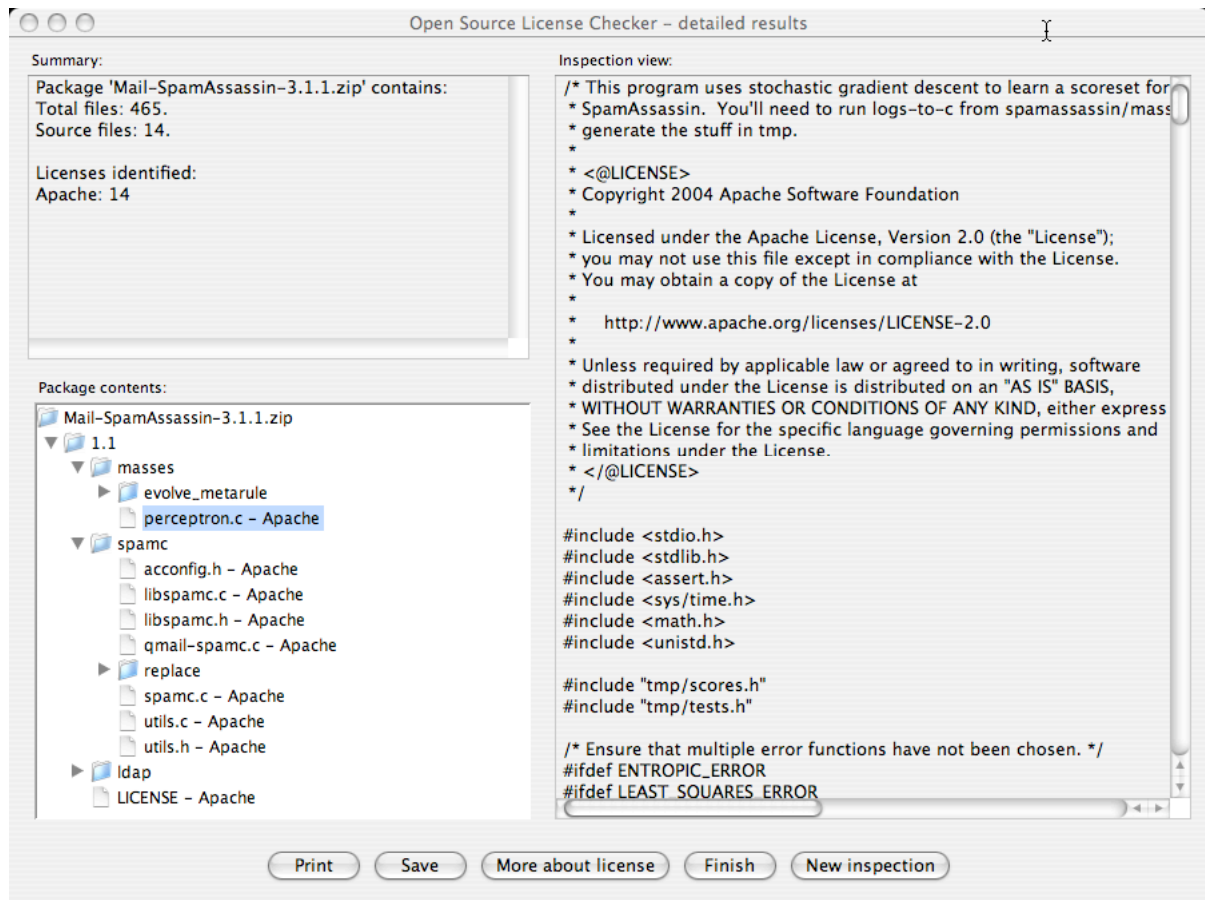
Screenshot 1. – Start-up screen

Appendix 2.



Screenshot 2. – Inspection Screen

Appendix 3.



Screenshot 3. – Detailed Results

Appendix 4.

Table 1. The Content of the Elements

Element	Content
Name	Contains the full name of the license, e.g. GNU General Public License.
Shortname	Contains the acronym of the license, e.g. GPL. This field is also used as the database key, i.e. every license must have a unique shortname.
Licensefile	Contains the name of the file containing the full license text.
shortlicensefile	Contains the name of the file containing the shortened license text, i.e. the one you usually add to the beginning of each source file.
compatibility	Contains three child elements, which contain the compatibility information of this license.
class1	Contains the shortnames of licenses that are fully compatible with this license.
class2	Contains the shortnames of licenses that are only compatible with this license if the licensed piece of software is dynamically linked as part of another software, i.e. static linking is not allowed.
class3	Contains the shortnames of licenses that are completely incompatible with this license.
keywords	Contains one or more 'group'-elements, which in turn contain one or more 'keyword'-elements. The 'keyword'-elements contain the (hopefully unique) keywords that are used to determine if the examined text matches with this license. Each keyword within a group must exist in order to generate a positive match, but a match from any one group is enough to generate a positive match
haropallo	This is an optional element and used as an indicator for internal entries in the license database. These entries have compatibility information, but no actual license data. For example, some reserved words that aren't supposed to appear in open source licenses are stored in the license database. Licenses with the 'haropallo'-element do not appear in the license browsing window.