The FOSSology Project

Robert Gobeille
Hewlett Packard Co.
bob.gobeille@hp.com

ABSTRACT

By its nature, the availability of FOSS has given computer scientists a large body of software and software projects to analyze. By having available source, version control system metadata, and open project communities, much can be learned about a software project, software development and collaborative project development. The goal of the FOSSology project is to create a public, open source software repository, together with tools to facilitate analysis, storage, and sharing of open source software and its metadata. FOSSology does license detection today.

Categories and Subject Descriptors

D.2.8 [**Software Engineering**]: Metrics; D.2.9 [**Software Engineering**]: Management *Copyrights*

General Terms: Legal Aspects

Keywords: license, analysis

1. INTRODUCTION

Some background is required to understand the motivation for creating the FOSSology project (http://fossology.org). In 2001 Hewlett Packard created an internal Open Source Review Board (OSRB) to examine each proposed use of open source released by the company. Release could be through open source software embedded in a device (printers, TV's, scanners, storage, etc.), linux distributions (Red Hat, Suse, Debian), or software in an open source project (like linuxCOE, FOSSology, and ~100 others). The OSRB performs due diligence to make sure that HP is honoring all the project open source licenses, and that any release of HP intellectual property is done purposefully and with the proper level of corporation approval.

The process for presenting a project to the OSRB starts with the submitter filling out a web form. On this form, one is required to enter, as free text, the names of all open source software used. For example, for a project that uses the apache web server, one might enter "httpd", "apache", "apache web server", "apache 2", or "apache 2.0.63". This example demonstrates three problems, a) how to resolve synonyms, b) how to relate a name to a specific set of code, and c) how to know if the submitter has entered all their software dependencies.

MSR'08, May 10–11, 2008, Leipzig, Germany.

The FOSSology project was born out of the idea that all three problems could be solved with a repository containing all the open source software used by a project brought to the OSRB, combined with tools to analyze that software for license terms and dependencies. With such a repository, the free text entry would be replaced by a selection process, and dependencies could be determined analytically. Of course, once such a repository existed, it could have other uses. Some that come to mind are identifying reused code (for missing license terms, reduction of code bloat, and a static analysis of origin), identifying projects within linux distros that don't meet the distro license guidelines [1, 2], tracking vulnerabilities, and in general being a place to store any file specific metadata. The term "file" is used literally here and includes isos, rpms, tars, .c files, etc. The FOSSology system, by default, recursively unpacks every container down to the indivisible file level. So when an iso contains a tar that contains an RPM that contains individual files, all levels (the iso, tar, RPM, and individual files) can all be analyzed and metadata attached to each.

2. The tangle of open source licensing

A Linux distribution (e.g. Debian Etch) has around 20,000 packages but upwards of 300,000 license declarations. Sometimes these declarations are simple references as in "this software is licensed under the GPLv2". In rare cases, license terms are negated, as in "this software is **not** under the GPL". Sometimes the entire text of a license is inserted into a file. But most commonly, licenses are adapted by taking a well known license (GPL, MPL, MIT, …) and changing a few words. This plethora of different licenses combined with the quantity of licenses, and the complexity of incompatible terms among licenses, makes for an open source governance legal headache. This is the reason why commercial companies like Black Duck Software (http://blackducksoftware.com) and Palimida (http://palamida.com) came to market.

2.1 An Example

Abiword is a popular project for those needing an open source solution to read Microsoft Word documents. Looking at the Abiword project in the Fedora 8 repository, one can see from the .spec file (used to create the Abiword RPM) that the project is under the GPLv2. This is the stated license by the project maintainers and will be recorded in the RPM. However, the FOSSology license analysis discovers 3,489 instances of 35 different licenses.

The following table is an abbreviated list of these discovered licenses. It is important to note that every file in the project was scanned and each file may contain 0-n licenses. The scan also includes container metadata, like that found in a gzip file, which could potentially contain a license.

License Count	License
2462	GPLv2 reference 5
264	LGPL v2.1+ GNU C Library exception
246	Phrase
236	FSF
51	LGPL v2.0 Reference
16	MPL contributor clause with dual license
7	zLib
3	X11
2	BSD UC Regents

Table 1. Abiword license frequency report. Abbreviated for illustration purposes, only 3,287 licenses in 9 categories are shown in the table, out of 3,489 licenses in 35 license categories in the complete abiword-2.4.6 listing.

When using the FOSSology web interface, one can click on each of the above licenses and a page of all the files discovered with that license will be displayed. Then you can click on any individual file and get a color coded display of what text actually matched what file.

The analysis results for each file are stored in the FOSSology database so that it can be used for subsequent research. This data includes a list that relates individual bytes of the file to the best matching license. This is how the web GUI can display the color coded matches. The "best match" is found through a genomics algorithm (bSAM) discussed later.

Most of the 3,489 references to licenses are as one would expect, GPL and GPL-like licenses. However, there are also two instances of the original BSD license (labeled "BSD UCRegents"). The original BSD UCRegents license included an attribution clause, and the GPL forbids additional license constraints. So here we have what looks like incompatible, and therefore nullifying licenses. But the license morass isn't that simple. The FOSSology tools identified two .cpp files that contained this BSD license. Manually looking at the makefile for those files shows that they are only used to build a spell library. Using ldd(1) to look at the compiled Abiword shows that this library is not linked and therefore, the licenses do not conflict. As an aside, when this was brought to the attention of the Abiword maintainers, they removed the dead code from the project, resolving the license confusion and the dead code in one stroke. This demonstrates a practical use of the FOSSology license analysis beyond legal issues.

2.2 Potential for Open Source Licensing to Jeopardize Corporate IP

Even though some open source licenses may be humorous, for example the Beer License and the WTF public license, they are also legal instruments which need to be honored. The Software Freedom Law Center recent lawsuits (http://www.softwarefreedom.org/news/2007/dec/07/busybox/) bear this out. There may also be more unintentional ramifications that could jeopardize corporate intellectual property. For example, the GPLv3 stipulates that if one contributes to a project by licensing a patent under GPLv3 "then the patent license you grant is automatically extended to all recipients of the covered work and works based on it". The potentially gives anyone who wants it, the right to use your patent (through a derivative work) even though the original intent was only to grant a license to a particular project.

3. FOSSology Architecture

The potential value of FOSSology is in how it may be used as a computing service and how results are shared. The FOSSology software itself is not particularly novel. It consists of a software repository, database, and a collection of tools. A script or web ui is used to load software into the repository. Agents are run by the FOSSology scheduler to extract component files from containers (iso's, tar, …), perform a specific analysis of each file (or metadata stored for the file), and store results. The results can be queried for reports or used by other analysis agents.

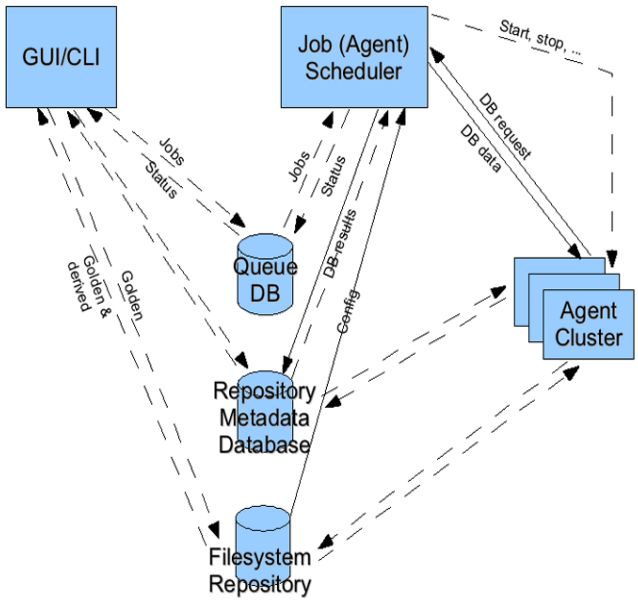

Figure 1 FOSSology overview

3.1 Filesystem Repository

Data is stored in either a filesystem repository or a database. The repository is the collection of files to be analyzed. The repository also provides a data cache for persistent intermediate data needed by the license analysis agent. Files are not stored under their original name or directory hierarchy. Instead, a filename derived from the sha1, md5 and file size is used. This ensures a unique filename, and creates a globally unique identifier (GUID). The GUID as filename ensures that each file is stored only once, even through the repository may be distributed across multiple hosts, each hosting files based on the GUID. A quick database query shows the benefit of storing files by GUID. In a sample of 5,216,426 files uploaded from Fedora 8 and Debian Etch, 1,095,773 (21%) were duplicates and therefore did not require added repository storage or further analysis. A database relation keeps track of the original file name, access control and place in the original file hierarchy.

4. bSAM - multifunctional pattern matching

One tool in FOSSology that deserves special mention is the pattern matching program. It uses an algorithm called the Symbolic Alignment Matrix (bSAM) [3]. bSAM is based on the Protein Alignment Matrix algorithm by Dayhoff, et al. [4,5], and provides FOSSology with a multipurpose pattern matching utility that researchers can use to perform different types of analysis. For example, license analysis is actually done by a pipeline of three agents. The first agent is a word tokenizing filter. The second agent is bSAM, which does pattern matching based on this tokenized version of the file, and writes match results (above a threshold parameter) to the database. Since tokens correspond to words and symbols in the file, the similarity and proximity of tokens can be used to determine similarity of text. A third agent in this license analysis pipeline simply removes the files that the first filter created.

bSAM can be applied to perform different types of syntax specific pattern matching by changing this first filter. For example, we are working on a code reuse agent that uses bSAM. For code reuse (or code originality), a filter tokenizes each file based on its file type and syntax. A C file may tokenize to something that looks like:

Myfunction ( ) { Var (String) ; for (Var = Num ; Var < Var [Num] ; etc.

Since SAM only compares tokens and the filter defines what the tokens are, bSAM can be used to find similar program structure and therefore a probability of code reuse. Using the GNU utility "objdump" to tokenize op codes we could even look for opcode reuse in binary files.

5. Yet another repository?

The idea of creating a large repository of software and metadata is not new. FLOSSmole (http://ossmole.sourceforge.net) has concentrated on analyzing metadata from SourceForge, Freshmeat, RubyForge, the FSF directory and others. Kenyon [6] has tools to facilitate the collection of some of the same data that FLOSSmole collects, but Kenyon also analyzes software and focuses on software evolution research. FLOSSmetrics (http://flossmetrics.org/) uses similar data as well as collects tools for source code analysis. Then there are commercial repositories like Krugle, Ohloh and Swik which also may make some of their data available and aggregate from all the above. Plus there are specific purpose repositories like that created by the Linux Foundation for analyzing LSB (Linux Standard Base) compliance, (see LSB Navigator http://ispras.linux-foundation.org/index.php/LSB_DB_Navigator).

So what does FOSSology.org have to offer? Our license analysis, based on several years of experience may be the best available. However, this is unsubstantiated. We hope that the availability of FOSSology.org will inspire someone to investigate this (measuring the number of licenses found in a known sample, number of false positives, number of false negatives). We also hope that as an active project, others will be motivated to make this analysis even better, as well as contribute other types of analysis. Through the building of a large open source repository we hope that FOSSology will prove as valuable of a resource to collect data as to analyze it.

6. The FOSSology community

We invite everyone to join the FOSSology community. One can use FOSSology to do their own analysis in two different ways. First, you can download the FOSSology code and populate your own repository. This is the most expensive and most flexible option. Owning the entire system allows one to access the database through direct connections and even bypass the scheduler to directly reference the repository files. However, populating a large repository can be very costly (in effort, time, and hardware). A second option is to use the public FOSSology database and repository. This saves one the hardware and effort to put together a repository but requires you to get permission to perform an ad hoc analysis. The agent must be vetted for use on the public repository because agents currently have wide permissions to insert, update, and delete from the repository and database.

7. Related Research and future work

We know that there needs to be additional methods for using the FOSSology public data. One clear need is to use the FOSSology repository and/or metadata as a repository of repositories [7]. With this option, one could export the parts of the repository and metadata that you are interested in and use it however you see fit. Becoming a repository of repositories means that there also needs to be a transfer protocol or data exchange format. TA-RE [8], a data exchange language, was proposed for exactly this use. The Open Archives Initiative (OAI) (http://openarchives.org) may be a better choice since it has a significant community backing. Our preference is for using simpler methods, for example, SQL upload files like FLOSSmole. How we will share data will require further research and input from those wishing to harvest our data.

Our simple solution to the file ambiguity problem, assigning a GUID based on file hashes and size, is a different approach for a similar problem that Mitre Corporation has been working on to enable automated security analysis. Their approach is a structured naming scheme, Common Platform Enumeration (CPE) (http://cpe.mitre.org/). One of CPE's goals is to create a naming hierarchy to delineate package relationships. In FOSSology, our approach is to use file analysis to determine relationships.

8. CONCLUSION

The approach we are taking to solve the problems mentioned in the introducion is to create a software analysis framework around a public software repository with a metadata database, and to develop it through an open source community effort. FOSSology.org was created for this purpose in December of 2007. It delivers code and documentation to enable one to set up their own infrastructure to facilitate code analysis but doesn't currently host a public software repository or database. We expect to add the public repository and database, hosted by the Linux Foundation (http://www.linux-foundation.org/), to FOSSology.org late February 2008.

9. ACKNOWLEDGMENTS

My thanks to Daniel German at the University of Victoria, British Columbia Canada. Without his help this paper would not have been written.

FOSSology is supported by Hewlett Packard, which is contributing full time staff as well as hardware (to the Linux Foundation) to support this project.

10. REFERENCES

[1] Free Software Foundation, “Licenses”, http://www.fsf.org/licensing/licenses/, accessed Feb. 19, 2008

[2] The Fedora Project , “Licensing”, http://fedoraproject.org/wiki/Licensing#SoftwareLicenses, accessed Feb. 18, 2008

[3] Krawetz, N., “Symbolic Alignment Matrix," http://fossology.org/docs/symbolic_alignment_matrix, 2008

[4] Dayhoff, M.O., Schwartz, R.M. and Orcutt, B.C., “A model of evolutionary change in proteins: matrices for detecting distant relationships.” In Atlas of protein sequence and structure, (Dayhoff, M.O., ed.), vol. 5, pp. 345-352. National Biomedical Research Foundation, Washington DC., 1978

[5] Schwartz, R.M. and Dayhoff, M.O., “Matrices for detecting distant relationships.” In Atlas of protein sequence and structure, (Dayhoff, M.O. ed.), vol 5 suppl 3, pp 353-358, National Biomed. Research Foundation, Washington DC., 1978

[6] Bevan, J., Whitehead Jr.,E.J., Kim S., Godfrey M., “Facilitating Software Evolution Research with Kenyon”, Proceedings of the 10th European software engineering conference held jointly with 13th ACM SIGSOFT international symposium on Foundations of software engineering, http://doi.acm.org/10.1145/1081706.1081736, 2005

[7] Sowe S.K., Angelis L., Stamelos I., Manolopou-los Y., “Using Repository of Repositories (RoRs) to Study the Growth of F/OSS Projects: A Meta- Analysis Research Approach”, Third International Conference on Open Source Systems. Limerick, Ireland, pp: 147-160., 2007

[8] S. Kim, et. al., “TA-RE: an exchange language for mining software repositories”, International Conference on Software Engineering. Proceedings of the 2006 international workshop on mining software repositories. http://portal.acm.org/citation.cfm?id=1137990, 2006