Lawful Software Engineering

Daniel M. German
Department of Computer Science
University of Victoria, Canada
dmg@uvic.ca

Jens H. Weber
Department of Computer Science
University of Victoria, Canada
jens@cs.uvic.ca

Massimiliano Di Penta
Deptartment of Engineering
University of Sannio, Italy
dipenta@unisannio.it

ABSTRACT

Legislation is constantly affecting the way in which software developers can create software systems, and deliver them to their users. This raises the need for methods and tools that support developers in the creation and re-distribution of software systems with the ability of properly coping with legal constraints. We conjecture that legal constraints are another dimension software analysts, architects and developers have to consider, making them an important area of future research in software engineering.

Categories and Subject Descriptors

K.5.m [**Computing Milieux**]: Legal Aspects of ComputingMiscellaneous; D.2.9 [**Software Engineering**]: Management—*Licensing*

General Terms

Legal Aspects

1. INTRODUCTION

Software is complex not only technically, but also from a legal point of view. Legislation places constraints in the way that software can be created, used, reused, integrated, and distributed.

Free, and Open Source Systems (FOSS), but also commercial software systems, are distributed under certain licenses, that determine under which condition the software can be used, modified, and re-distributed. Local and international legislation places constraints in the way a software system can be deployed and used.

At the minimum, developers should be (i) aware of constraints that licenses impose when they decide to integrate somebody else's software into their own, and (ii) aware of the limitations that legislation, such as privacy law, imposes on what a software system can and cannot do.

Software systems are intrinsically complex, and subject to continuous changes. So is the law and software licenses. For this reason, software developers are confronted often with problems whose solutions require both technical and legal expertise:

- Can I reuse source code of an existing software system, and if yes, under which (legal) conditions?
- What architecture should I choose to integrate components having certain licenses?
- I'm going to integrate a component for which I don't have the source code. Is there any way to determine how was it source code licensed, and whether or not I could use the component under certain conditions?
- Are there any laws that could restrict the way the software operates?

Similarly to how previous research work proposed approaches to support developers in tasks such as change impact analysis, identification of source code vulnerabilities or design smells, we believe that there is a strong need for techniques and tools that support developers in coping with legal issues from a technical point of view. In this paper, we argue that "lawful software engineering" is an important future challenge for the software engineering research community.

2. INTELLECTUAL PROPERTY AND SOFTWARE ENGINEERING

The main types of intellectual property that affect software engineering are patents, copyright. On the one hand, patents protect inventions, and they are monopolistic by nature: the patent owner has a monopoly in its exploitation for up to 20 years. In recent years software and business methods patents have been an area has received significant scrutiny due to the difficulty of implementing a legal regime that is fair to creators, users, and society [21, 1], with some arguing that they should be eliminated altogether [18].

On the other hand, copyright protects the expression of an idea. Like patents, it gives a monopoly to its owner (for a period of at least 70 years in the United States [24, Ch.3]). Copyright provides several protections to its owner, but two of the most important, from a software engineering point of view, are the right to make copies of the software (e.g., copy verbatim source code), and the right to make derivative works (e.g., using a software product as the basis of a more complex one). Licenses are the legal mechanisms that the intellectual property owner can use to allow a third party to exploit any of these rights.

The current patent systems regime (particularly that of the United States) makes it difficult for a software developer to determine, in advance, if her software requires a license from the patent holder. Because copyright protects the expression, many software programs can implement the

FoSER 2010, November 7–8, 2010, Santa Fe, New Mexico, USA.

same idea without violating the copyright of each other (the way many authors can write different stories of spies).

One of the goals of software engineering principles—and especially of good design principles—is to promote reuse. Most large software applications are not built from scratch; they are built by combining several components such as reused code snippets, self-contained binary libraries, or other applications. Over the last decade, various research efforts have focused on the technical aspects of supporting and improving component-based software development processes. For example, Garlan *et al.* discuss the challenges of component development due to architecture and interface mismatches [7]. Typically, developers would choose a potential software to reuse (such as libraries), and request to the management to negotiate a license to the desired component. This license would specify, at the very least, the rights and responsibilities of the copyright owner (the licensee) of the work and its user (the licensee). A license typically covers issues such as warranty, liability, and remedies in case of one of the parties failing to obey the license, and are typically granted via a contract (although there is some argument that some FOSS licenses are not contracts, such as the General Public License, see [17]).

Arguably, the advent of free and open source software (FOSS) has created an ecosystem where software naturally occurs. Many commercial and governmental organizations are part of this ecosystem, where they produce and/or reuse FOSS software as part of their typical operations.

Three recent developments have complicated licenses, and how they are acquired:

- a recent legal shift in the view that software producers should deliver free-of-defects software;
- the global nature of software development; and
- the growing availability of FOSS.

3. PERFECT SOFTWARE? ON WARRANTY AND LIABILITY

The software industry has traditionally operated different from other industry sectors when it comes to warranty and liability concerns. A typical license agreement stipulates that the software product is provided 'as is' without any express or implied warranty, and that the vendor shall in no event be held liable for any damages arising from the use of the software. This contractual practice has increasingly been challenged. Last year, the American Law Institute (ALI) unanimously approved a report on the "Principles of the Law of Software Contracts" [3]. In their introduction to the report, the editors state, "perhaps no other commercial subject matter is in greater need of harmonization and clarification." Principle 3.05 states that: "A transferor that receives money or a right to payment of a monetary obligation in exchange for the software warrants to any party in the normal chain of distribution that the software contains no material hidden defects of which the transferor was aware at the time of the transfer. This warranty may not be excluded." In other words, the Principles suggest that the software vendor has a non-disclaimable obligation to deliver software that is free of material defects.

The Principles (particularly 3.05) have drawn significant criticism by representatives of the software industry, as well as business and legal experts. In a rare alliance, Microsoft and the Linux Foundation have sent a joint letter to ALI asking them to reconsider[4]. Lawyers and business experts have expressed concerns about negative impacts on the software industry in terms of flexibility, cost and uncertainty. In his article "Flawed ALI Software Contract Principles", Nimmer criticizes, among other points, that it is not clear what the words "material" and "hidden" defect mean [15].

While the ALI Principles are not law but a recommendation, it has obviously been drafted with the intent to influence lawmakers. Software organizations have begun to revise contracts to adhere to the Principles. Mark Radcliffe (Linux Foundation) published a list of recommendations on how to implement the Principles, including a method for disclosing material bugs and a review of implied warranties. ("This issue is particularly important for the implied indemnity for intellectual property infringement") [16].

While the ALI Principles have been issued in the U.S. context, similar discussions are happening in other jurisdictions around the world. It is clear that the software industry will be facing significant challenges driven by changing legal and regulatory frameworks throughout the world. A patchwork of privacy regulations and emerging industry specific regulations, such as the Canadian medical devices regulation, which now covers all electronic health record software systems, are further complicating the task of engineering "valid" software. Moreover, cloud-based software systems often cross-cuts the boundaries of regulatory frameworks—a fact that further increases the engineering challenge.

4. THE GLOBAL NATURE OF SOFTWARE DEVELOPMENT AND USAGE

Different types of intellectual property have different international protections. For example, copyright is globally protected (there are small differences between countries laws, e.g., the Canadian Copyright Act does not define the right to make available, nor digital rights management, while the American Act does), but patent and trademark law is national in scope.

4.1 Where are you and your users?

An organization in one country might be able to use, without paying royalties, a patent if they are outside the jurisdiction of its owner. However, the company might be held liable for infringement if its starts to sell its services or products in such jurisdiction. Research In Motion (RIM), is perhaps the best example of this risk to the computer and software industry. RIM was sued for patent infringement by NTP Inc. (NTP Inc. v. Research in Motion, LTD, 03-1615 (Fed. Circ. Dec. 14, 2004). One of RIM defenses was to claim that its processing (and potential patent infringement) occurred in Canada and hence was beyond the jurisdiction of the United States. The Federal Court panel had a different opinion and ruled that "The location of RIM's customers and their purchase of the Blackberry devices establishing control and beneficial use of the Blackberry system within the United States satisfactorily establish territoriality under Section 271 (a)[...]". This ruling raised important issues of territoriality to the point that it prompted the Canadian Government to fill an *amicus curae* supporting RIM, where it stated that "The panel's adoption of this 'control and beneficial' use rule also raises the risk that Section 271 (a) might be accorded inappropriate extraterritorial application, contrary to the principles of comity affecting Canada and the United States"[13] Eventually, after RIM exhausted all its legal options, it payed $612.5 million to settle the lawsuit.[20].

4.2 If we disagree, where do we hold a trial?

An important aspect of a contract is defining the jurisdiction that would be used to resolve any conflict between the parties involved. Few years ago, the Faculty of Engineering of the University of Victoria[1] decided to open-source its Web-based Recruiting System Mamook[2]. The University administration, however, deemed the Open Source Initiative (OSI) approved open source licenses[3] inadequate for the purpose, a decision that initiated a process that resulted in the University of Victoria authoring a new open source license that was eventually approved by OSI. One of the main concerns of the University administration was that the jurisdiction of the license should be British Columbia (at that time no OSI-approved license allowed such a clause). Eventually OSI approved the *Adaptive Public License*[4] that allows the licensor to indicate the governing jurisdiction.

4.3 Sensitive data cross jurisdictional borders?

In 2006, there was a growing concern in Canada that subcontracting medical information processing to the United States could expose individuals sensitive information to the PATRIOT Act. In response, the Government of Canada issued a report outlining the privacy concerns of "transborder data flows", in particular when the government sends private and sensitive information about individuals to other countries as part of outsourcing. The main outcome of the report is that transborder data flows is there to stay, but that there is a need for current privacy best practices to become more uniform throughout the federal government and for additional measures to build upon and complement the existing safeguards [22].

5. USE THE (OPEN) SOURCE, LUKE!

The advent of free and open source software (FOSS) has created a large ecosystem of applications, libraries and components that are readily available for download and usage. The intellectual property of FOSS is protected by licensing mechanisms and copyright notices that determine how an open source can be used, even impacting the architecture of a system [10].

Dealing with issues related to FOSS licensing is not trivial, as at the time of writing there are 65 open source licenses approved by the OSI, and many more in use; each of them imposing particular constraints concerning how one can use and/or change a software. In fact, identifying the license of a single source code file is not trivial[12, 11].

In [8], German *et al.* demonstrated that license auditing is a complex and difficult process. The large set of existing licenses, the constraints they pose, and the complex dependency relationships among packages determine a situation in which it is difficult to understand under what conditions a package can be used and/or redistributed. Using an empirical study they showed that a deeper understanding of licensing issues requires expertise, and is need of automation (at least partially).

There are two main ways in which FOSS can be reused: as a component, and by copying source code from one software system to another.

[1]Two of the authors belong to the University of Victoria
[2]`http://sourceforge.net/projects/mamook`
[3]`http://www.opensource.org/licenses`
[4]`http://www.opensource.org/licenses/apl1.0.php`

5.1 Reusing FOSS components

Using a FOSS component as part of another one poses several challenges:

What is the license of the component? As demonstrated in [8] this is not trivial to determine, but it is a responsibility of the people using the component to make sure that they do it lawfully[17]. The problem is even more complex when the component's source code is not available. In such a case, a viable solution would be to decompile the source code and use the information it contains to query Web code searching repositories like Google Code Search[5] or Koders[6] to identify the component's license [5].

Licenses mismatch between components. In [10] showed that Bugzilla is created with components that use 10 difference licenses. It is therefore important to know how licenses can (or cannot) interact between each other [2].

How are the components connected? The method of interconnection between components might restrict the integrator. For example, the Free Software Foundation (FSF) lists "GPL-incompatible" licenses; in their view, any program under such licenses (such as the Eclipse Public License, or the Mozilla Public License) cannot link to a GPL licensed library. [23] proposed a method to determine the way some components are connected by instrumenting a C compiler.

Tracking the evolution of the license of the component. When the license of the component changes, the licensee might not be able to continue using it. This is a growing concern as many licenses are been updated[6].

5.2 Copying FOSS

Copying of source code across FOSS is well documented [14, 9]. Sojer and Henkel interviewed several hundred developers and discovered that copying FOSS code by commercial enterprises is now common, but in many cases their software developers lack understanding of the legal risks associated with this activity and their organizations lack policies to guide them[19]. As described in [9], embedding source code from another copyright owner creates several challenges, such as:

- Tracking the copyright owner and license of the copied code.
- Determining if the license of copied code is compatible with the intended use of the code.
- Managing the evolution of the copy with respect to the original is difficult.

6. THE FUTURE

As outlined in this document, legislation affects in many manners the way that organizations (and their software developers) create, deliver and operate software systems. The last three years have seen a healthy growth in research in this field, as exemplified by the Requirements Engineering And Law Workshop (started in 2008) and the inclusion of a "Licensing" session at ICSE this year (2010). We believe the following areas will be worthwhile of future research investigation:

Improving software development models and methods to include legal issues. As described above, the impact of the law is undeniable in how software is built. As a consequence, models and methods should now incorporate these concerns as an important part of the software devel-

[5]http://www.google.com/codesearch
[6]http://www.koders.com/

opment process. In particular: how to handle FOSS as part of a system (either used as a component, or when the code is copied), how to handle restrictions in processing and data exchange imposed by legislation (such as privacy laws), and how jurisdictional issues might affect the creation, use and deployment of a software system.

Educating developers, lawyers, judges and the public in general. The ACM Computing Curriculum includes intellectual property as one of its topics, but needs to be expanded to include the implications of using FOSS, how privacy and other legislation effects what a software system can and cannot do, or how it should do it, and how jurisdictional issues affect global software development (and deployment). The software engineering research community should also be concerned on how to educate the public, lawyers, and end users on these issues.

Auditing methods for intellectual property evaluation and compliance. This is important for those who create the software (are my software developers complying with legal issues?), for those who buy software created by others (is the product I am buying privacy compliant, intellectual property compliant, and in general law compliant?) and for those wanting to acquire organizations whose assets include software (who owns the code? if needed, does the organization have a proper license to use it?).

Continuing with the creation of tools to assist software developers and auditors in their evaluation of licensing compliance of a software system. This is a very underdeveloped area of research; currently this evaluation is typically done manually by experts.

In conclusion, researchers should aim towards work whose goal is to assist developers in the creation of lawful software systems.

7. REFERENCES

[1] P. S. Abril and R. Plant. The patent holder's dilemma: buy, sell, or troll? *Commun. ACM*, 50(1):36–44, 2007.

[2] T. A. Alspaugh, H. U. Asuncion, and W. Scacchi. Intellectual property rights requirements for heterogeneously-licensed systems. *Int. Conf on Requirements Engineering*, 0:24–33, 2009.

[3] American Law Institute. Principles of the Law of Software Contracts. 1PLSCOT, 2010.

[4] K. Copenhaver and H. Gutierrez. Re: Principles of the Law of Software Contracts. Linux Foundation and Microsoft Corporation `http://microsoftontheissues.com/downloads/Microsoft-LinuxFoundation-Letter.pdf`, May 2009.

[5] M. Di Penta, D. M. German, and G. Antoniol. Identifying licensing of jar archives using a code-search approach. In *Proc. of the 7th Working Conference on Mining Software Repositories, MSR 2010*, 2010.

[6] M. Di Penta, D. M. German, Y.-G. Guéhéneuc, and G. Antoniol. An exploratory study of the evolution of software licensing. In *Proc. of the 32rd International Conference on Software Engineering (ICSE'10)*, pages 145–154, 2010.

[7] D. Garlan, R. Allen, and J. Ockerbloom. Architectural Mismatch or Why It's Hard to Build Systems Out Of Existing Parts. In *ICSE*, pages 179–185, 1995.

[8] D. M. German, M. Di Penta, and J. Davis. Understanding and auditing the licensing of open source software distributions. In *18th Int. Conf. on Program Comprehension (ICPC'2010)*, pages 84–93, May 2010.

[9] D. M. German, M. Di Penta, Y.-G. Guéhéneuc, and G. Antoniol. Code siblings: Technical and legal implications. In *Proc. of the 2009 Working Conference on Mining Software Repositories, MSR 2009*, pages 81–90, 2009.

[10] D. M. Germán and A. E. Hassan. License integration patterns: Addressing license mismatches in component-based development. In *31st Int. Conf. on Software Engineering, ICSE*, pages 188–198, 2009.

[11] D. M. German, Y. Manabe, and K. Inoue. A sentence-matching method for automatic license identification of source code files. In *25nd IEEE/ACM International Conference on Automated Software Engineering (ASE 2010)*, 2010. To be presented.

[12] R. Gobeille. The FOSSology project. In *MSR '08: Proceedings of the 2008 International Conference on Mining Software Repositories*, pages 47–50, 2008.

[13] Government of Canada. Brief Amicus Curiae of the Government of Canada in Support of the Request for Rehearing En Blanc made in the Combined Petition By Research in Motion, LTD. for Pannel Rehearing and Rehearing En Blanc. 03-1615 in the United States Court of Appeals for the Federal Circuit, Jan 2005.

[14] J. Krinke, N. Gold, Y. Jia, and D. Binkley. Cloning and copying between gnome projects. In *MSR*, pages 98–101, 2010.

[15] R. Nimmer. Flawed ALI Software Contract "Principles". `http://www.ipinfoblog.com/archives/licensing-law-issues-flawed-ali-software-contract-\principles.html`, May 2009.

[16] M. Radcliffe. Dealing with the Flaws: Principles of Software Contracts. Linux Foundation `www.linuxfoundation.org/publications/radcliffe_ali_alert.pdf`, May 2009.

[17] L. Rosen. *Open Source Licensing: Software Freedom and Intellectual Property Law*. Prentice Hall, 2004.

[18] P. Samuelson. Software patents and the metaphysics of section 271(f). *Commun. ACM*, 50(6):15–19, 2007.

[19] M. Sojer and J. Henkel. License Risks from Ad-Hoc Reuse of Code from the Internet: An Empirical Investigation. Preprint, available at `http://papers.ssrn.com/sol3/papers.cfm?abstract_id=1594641`.

[20] A. R. Sorkin. Research In Motion Settles Patent Suit. Legal Column, New York Times, March 2006.

[21] M. E. Thatcher and D. E. Pingry. [Software patents] The good, the bad, and the messy. *Commun. ACM*, 50(10):47–52, 2007.

[22] Treasury Board of Canada Secretariat. Privacy Matters: The Federal Strategy to Address Concerns About the USA PATRIOT Act and Transborder Data Flows. Catalogue No. BT22-104/2005E-PDF, 2006.

[23] T. Tuunanen, J. Koskinen, and T. Kärkkäinen. Automated software license analysis. *Automated Software Eng.*, 16(3-4):455–490, 2009.

[24] United States Copyright Office. Circular 92 Copyright Law of the United States of America and Related Laws Contained in Title 17 of the United States Code, June 2003.