

Open Source Collaboration in Higher Education

GUIDELINES AND REPORT OF THE LICENSING AND POLICY
SUMMIT FOR SOFTWARE SHARING IN HIGHER EDUCATION

VERSION 1.0

MARCH 16, 2007

CONVENED BY:

DANIEL GREENSTEIN
ASSOC VICE PROVOST, SCHOLARLY INFORMATION
UNIVERSITY OF CALIFORNIA

BRAD WHEELER
CHIEF INFORMATION OFFICER
INDIANA UNIVERSITY

© 2007 Trustees of Indiana University
Creative Commons Attribution License 3.0

FOREWORD

Higher education has long valued the collaborative production and sharing of knowledge. Thus, the rise of software communities powered by the near frictionless cost of the Internet and open source software production techniques seems a natural fit for colleges and universities. One vexing challenge, however, has been finding a common legal and policy framework for software contributions, licensing, and distribution of collaboratively developed work. In the absence of a common framework, heterogeneous policies and licenses will remain an unhealthy drag on considerable economies that we can harness when we benefit from others' investments. Colleges and universities, as the primary beneficiary of software sharing, must be proactive in creating, adopting, and advocating for a common framework if we are to ever take full advantage of these opportunities.

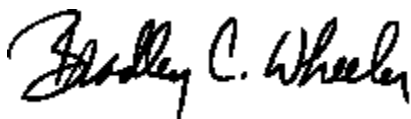
We are grateful that the Andrew W. Mellon Foundation recognizes this problem and funded the October 2006 gathering of some thought leaders to work on these challenges. Our distinguished attendees were drawn from university legal counsel, technology transfer offices, open source projects, governmental funding agencies, and foundations, with representatives from multiple continents. We worked hard via electronic communication before the summit, during two days of intensive face-to-face discussions, and through much follow up afterward. The electronic communications were open to all at <http://collabtools.org>.

This document is one of the work products of the summit, and it includes a set of educational materials for an institution's engagement with open source application software. Action and implementation will be the key arbiter of value, and we urge institutions, projects, and funding agencies to engage in the ongoing work of refining and implementing the work started here. We are thankful to all who took their time to participate and to Tina Howard, who carried the real work of writing and editing these materials.

Sincerely,



Daniel Greenstein
University of California Office of the President



Brad Wheeler
Indiana University

TOPICS

	Foreword	i
1.0	Introduction	1
2.0	Summit Recommendations and Licensing Roadmap	2
3.0	Benefits of Open Source Software To Higher Education	3
4.0	General Questions	3
5.0	Agreements and Licenses	5
6.0	Patents and Third-Party Patent Licenses	12
7.0	Developer Issues	12
8.0	Patents and Copyrights Outside of the United States	15
	Appendix A. Summit Participants	19

1.0 INTRODUCTION

Open source projects are becoming more common in the higher education community and have shown promise as a cost-effective means of supplying software that enhances the community's ability to serve its educational mission. Projects such as Sakai, Quali, Chandler, Open Source Portfolio, Fedora and DSpace are examples of how institutions can leverage the considerable investments they make in essential enterprise-wide systems. These shared investments can lower institutional life cycle costs while improving system fit and quality.

Even with continued recognition of the benefits of open source licensing, there are still questions to be answered about how the higher education community can facilitate collaborative open source projects that serve community interests. Policies and infrastructure regarding the licensing of intellectual property created by the universities differ by institution, and are still evolving. But at the same time it is critical that users and the community as a whole be able to expect all contributors to grant the same set of rights with respect to their contributions.

The state of open source software among institutions of higher education is characterized by:

- The lack of a uniform license (or contributor agreement) that will apply to all contributions to open source projects by and for the higher education community,
- The need to minimize the number and variety of open source licenses that institutions wishing to adopt open source software created for the community need to review and understand, and
- The need to avoid repeated and potentially time-consuming negotiations among institutions in the context of each new project.
- Although the Educational Community License (ECL) was created for higher education, it can be improved to better serve the needs of the community.

To address these issues, a representative group of delegates came together in October 2006 for a summit titled "Licensing and Policy Summit for Software Sharing in Higher Education."¹ Although the group was tasked with addressing several topics, most of the discussion over the course of the Summit centered around licensing issues, specifically:

- Creating or recommending a common form of open source license that addresses both copyrights and patents, and

¹A list of Summit participants is in Appendix A.

- Working toward a uniform agreement among contributors that will address both copyright and patents and the unique issues faced by universities in connection with funded research, as well as balance the rights of individual inventors within the institution with the desire of the institution to make licensing decisions that will advance higher education generally.

While some issues remain outstanding and the discussion continues, the group did come to agreement on some important points.

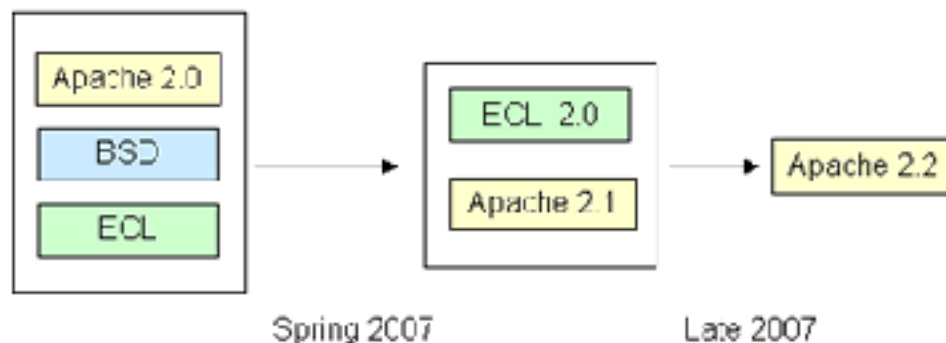
This document is intended to provide answers to some basic questions regarding these issues, as well as explore the questions that remain to be answered. The information included in this document is specific to higher education but may provide a useful starting point for those in other sectors who are or wish to become involved in collaborative inter-institutional open source projects.²

The next section summarizes the Summit recommendations, with more detail on the discussion provided throughout the document. The remainder of the document explains the benefits of open source to higher education followed by an overview and list of questions related to legal issues, specifically licensing and patents. The final section addresses laws outside of the United States.

2.0 SUMMIT RECOMMENDATIONS AND LICENSING ROADMAP

The consensus of the Summit group was to create a new version of the ECL in the short-term and to work with the Apache Software Foundation to incorporate desired features into its next license release (see Figure 1). The primary points of consensus among the Summit group were:

1. A common licensing framework is preferred.
2. A simple license is preferred, but it must have a patent clause to meet the needs of institutions that grant third-party patent licenses. We decided on a license that has a patent clause. We want a common licensing framework.



²For a definition of Open Source <http://www.opensource.org/docs/definition.php>.

Figure 1. Licensing roadmap after Summit recommendations

3. The ECL 2.0 created as a product of the Summit is an evolution of the Apache license, not the original ECL. There was considerable interest in the Apache licenses to further consistency among open source projects, but it could not be fully supported for universities because its patent license was considered too broad (some universities could not by policy agree to such broad language). One question asked was whether there is value in having a separate license for higher education. The answer was no at this time, but that any existing license must meet the needs of higher education for broad adoption to occur.
4. The new ECL contains Apache 2.1 language with necessary modifications for colleges and universities. The hope is that the ECL 2.0 and Apache 2.1 can ultimately be merged.

3.0 BENEFITS OF OPEN SOURCE SOFTWARE TO HIGHER EDUCATION

The higher education community can benefit from open source software in many ways.

- Share resources - When multiple institutions are involved in developing a product, they share resources that no single institution could provide. Multiple inputs and approaches also result in an improved product.
- Faster repairs and high code quality - Open source products -- even commercial ones -- are often more reliable than products with closed source code because everything is continually peer-reviewed through use of the software. If something doesn't work, it is fixed immediately and distributed rather than being fixed in the next release.
Further, broad adoption across the community increases code quality because more people are involved in using the code.
- Build a lasting community through broad sharing of ideas and resources - We have the opportunity to create a framework for sharing software within the community, while at the same time avoiding unnecessary transaction and legal costs with a single form of license.
- Realize a total cost advantage by utilizing shared resources for development and maintenance. Collaborative products also are more sustainable over time.
- Motivate commercial support - With the rise of open source software, the number of sources of commercial support has also grown. Help is generally easy to find and less costly than support and maintenance fees paid to software vendors.
- Empower adopters/users to gain the benefits of using shared software while also allowing them to easily make modifications to meet their unique needs.

4.0 GENERAL QUESTIONS

Aren't off-the-shelf software packages better than collaborative open source software projects?

With any software package, the quality depends upon several factors. Both open and closed source products can be either good or bad. The quality of an open source product often depends on the size and quality of the developer community involved. Large, collaborative open source products have proven themselves to be equally or more reliable than off-the-shelf products. Further, one advantage of open source is that when there is a problem, it can be fixed and redistributed immediately.

How does the cost of open source software collaboration compare with the cost of custom software packages?

The up-front cost of collaboration on a large software system may seem high, but the benefits in the end can make it a better proposal in many cases. When universities work together to meet a common need, several benefits are realized.

- The software is created with the needs of higher education institutions already understood. This is true especially when the community has longstanding experience with these applications, and problems or areas for improvement of existing applications are used as a starting point.
- Shared resources spread the cost over many universities instead of multiple universities paying the same cost for the same functionality.
- Institutions can leverage their relationships and their strengths to create a common system.
- Updates and customization can be done in-house when they're needed.
- When the source code is open, there are frequent updates available, and applying these updates costs far less than purchasing new versions of commercial software that frequently involve only minor improvements.
- The availability of a shared code base can lower the barrier to entry of commercial vendors who wish to build new products, or add new features, using that code base.

The benefits are even greater for smaller institutions or institutions with relatively lower levels of resources because they are able to scale their contribution to their resources and still use the final product or package.

Are there specific foundations or organizations I should be aware of if I'm interested in open source software projects?

There are several organizations that either support or create open source products. However two organizations have been created by joint efforts to directly support the higher education community with multiple open source

projects: Kuali and Sakai. Both are funded by several universities (among other organizations) and are run by boards of directors.

- Kuali Foundation (www.kuali.org) supports the development and maintenance of open source administrative software for Carnegie Class institutions. Its first project was a financial system and it has now launched a research administration system project. Kuali products are developed using the Community Source development model, an open source model with more clearly defined roles and financial commitments.
- Sakai Foundation (www.sakaiproject.org) supports development of an online collaboration and learning environment used to support teaching and group collaboration, including such tools as schedule, chat rooms, message center, and wiki. The Open Source Portfolio Initiative (www.osportfolio.org) is also part of the Sakai project. Sakai is a free and open source product built and maintained by its users and other members of the Sakai community.

Outside of the United States, open source projects are supported by a combination of university, government and community organizations. Many universities have established offices that specifically handle open source projects and issues.

Who is responsible for general open source decisions?

In the United States, the Open Source Initiative (www.opensource.org) promotes open source technologies and offers certification for open source licenses and software. Although not legally required, this certification indicates that a license or product complies with OSI's definition of open source (www.opensource.org/docs/definition.html).

On an institutional level, a technology office can provide resources and guidelines. University counsel may also be involved in reviewing, negotiating and approving licenses and agreements. Chief Technology Officers and Chief Financial Officers also play an important role in influencing university policy and advocating for open source collaborations.

5.0 AGREEMENTS AND LICENSES

Some of the questions that typically arise in connection with open source projects relate to the legal issues: What are my responsibilities as a developer or an institution that has staff members who are contributing code? What form of open source license should be used? Can software that is subject to other forms of open source licenses be used in connection with a project?

There are several key points to consider, and they are addressed in this section. First we will discuss the contribution agreements by which contributors provide permission for their code to be used in connection with a project, and the various forms of open source license that can be granted to users as a result. Second,

we will discuss the role of software patents in open source licensing. Finally, we will address some questions specific to developers.

General Overview

The software licenses granted by those holding rights in software describe and define how their code may be used by others.

A general principle is that there must be no incompatibility between the terms of the rights granted by contributors in their contribution agreements and the rights that can be granted to users (outbound license). Another principle is that licensing should be as friction-free as possible, and for that reason, well-known open source licenses and uniform forms of contribution agreement should be used as much as possible.

Two of the main goals of the Summit were to identify an open source license that could be used consistently across the higher education community and to establish a uniform contribution agreement that would be compatible with the open source license. Another major goal of the Summit was to make sure that the forms of agreement would address patents, and not just the copyright in the software.

What is different about open source licenses?

Open source software generally refers to software that is made available in the form of source code that is readable and modifiable by users, and under a license that allows users to:

- access, install and run the software for any purpose at no cost,
- modify the original software, and
- redistribute either the original or modified software.

By contrast, commercially developed software usually -- but not always -- maintains closed code, requires a licensing fee and either does not allow modifications or does not support modified software.

What licenses are available?

While there are a large number of free and open source licenses that can be used to publish open source code, there are three primary categories of open source licenses.

- Licenses that give permission to use, copy, modify and share code for any purpose, but require that the same license terms be applied in the event of any modification or redistribution of the original code (often called “copyleft” licenses). These licenses may be incompatible with other forms of open source licenses that allow more freedom with respect to how modifications of the

original code can be distributed, or with use of the software in conjunction with proprietary software.

- Licenses that are intended to allow the software to be used in connection with projects that may use a different open source license, or to be used in connection with proprietary software, but that are also intended to maintain the user's freedom to use, copy, modify and share the original open source library or other software. This type of license, often called "weak copyleft," is frequently used where the open source software is a library that is intended to be used by several types of programs.
- Licenses which impose no significant restrictions on redistribution or the license under which software may be redistributed other than maintaining the copyright notice and disclaimer. These are often called "permissive" or "academic" licenses.

The first type of license is referred to as a "GPL-type" license, for the popular General Public License. The second includes the LGPL ("library" or "lesser" GPL) and the Mozilla license, named after the project that published the mostly commonly used version of this type of license. The third is often referred to in educational circles as a "BSD-type" license for the Berkeley Software Development license. In many cases, open source project will adopt these licenses without any changes, but there are variations of all of the types.

One of the most commonly used licenses is known as the Apache 2.0 license, used by the Apache Software Foundation. The Apache 2.0 license can be viewed as an evolved form of the BSD-type license. It provides more detail and clarity than previous BSD-type licenses, which tended to be extremely brief, and it includes a patent license as well as a copyright license.

The Educational Community License (ECL) is a license that has been developed specifically for the higher education community. One of the outcomes of the Summit was a new version of the ECL, the ECL 2.0, which is recommended for the higher education community. It is currently being OSI certified and will be released when the process is completed.

Historically, universities developed their own software distribution licenses. The original ECL was created to provide consistency. Although the philosophical intent of ECL was to be a BSD-type license, the brevity of the BSD-type licenses also may leave room for ambiguity, and the ECL did not include a patent license.

The consensus of the Summit group was to create a new version of the ECL by adding a patent clause. This was done to address the concerns of universities that often grant patent licenses to third parties. Although Apache 2.0 (not the original ECL) was the basis for the new ECL, the group believed that a separate license was necessary to ensure that the unique needs of the higher education community continue to be met. The Apache Software Foundation works with various industries and groups

worldwide. A group of higher education representatives will work with Apache, which was represented at the Summit, to inform the work on the next version of the Apache license and the hope is that the license that emerges will be well-suited to the needs of higher education.

What are contributor agreements and “inbound” and “outbound” licenses?

There are three types of agreements in the licensing framework of a typical open source project: contributor agreements, inbound licenses and outbound licenses.

Contributor agreements refer to the agreements that institutions and individual contributors must sign to contribute code to an open source project. They may assign ownership of the copyright in the contribution to the project, with or without a license back to the original developer for use in other projects, or they may grant a license to use, modify and redistribute the contribution. Contributor agreements are, in practice, a form of inbound license.

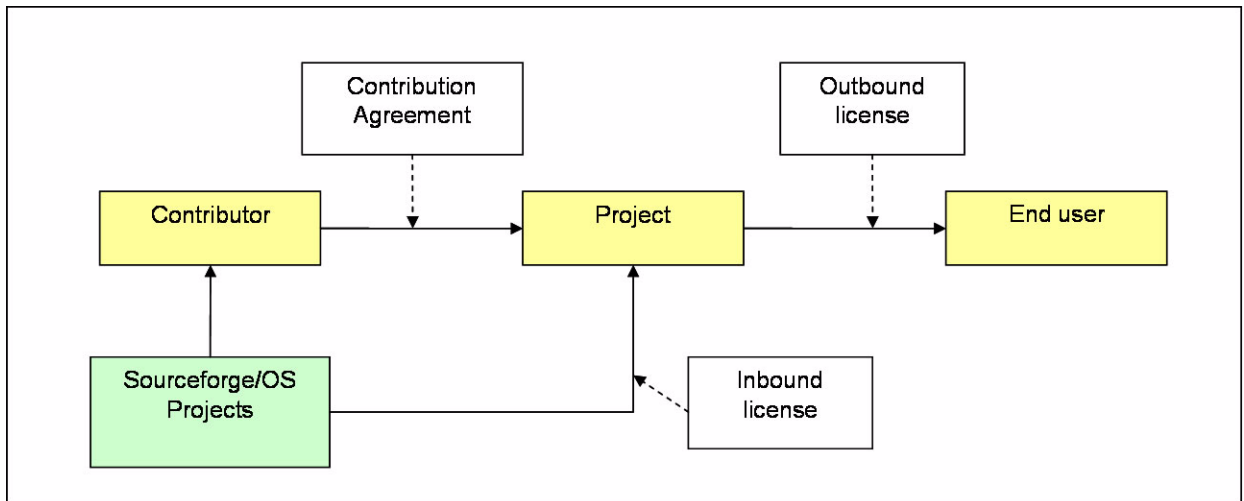


Figure 2. Standard open source project licensing framework

“Inbound” license refers to the license that applies to software that is coming *in* to the project. This may include software that is contributed directly to a project, but it may also include open source software from other sources that is to be included in the distribution.

“Outbound” license refers to the license granted to users in connection with the release of a product. Note that a single distribution may include software from a variety of sources, some of which was developed specifically for the project and some of which was created independently. While the project as a whole may be bundled under a particular outbound license, software that was created independently that is included within the distribution will typically remain subject to the terms of the outbound license under which it was originally released.

Who signs and what terms typically govern Contributor Agreements?

Contributor Agreements typically are required when becoming involved in an open source software project. Both individual developers/contributors, and the institutions with whom these individual developers/contributors are affiliated will typically be asked to sign a contributor agreement, because otherwise the open source project would need to know the policy at each contributor institution regarding whether individuals or the institution controls the rights to the intellectual property developed by those individuals.

A contributor agreement will typically include a permanent, non-exclusive license to the contributed software, representations that the contributor is the author of the contributed software, and has identified any software written by others that is being contributed, and disclaimers of liability to protect the contributor in the event the contributed software does not function as intended.

In some cases, a contributor agreement will provide for an assignment of ownership of the copyright in the contribution (perhaps with a license back to the original contributor if the contribution is not intended to be used exclusively by the open source project). The reason for this is that it can be difficult for an open source organization that has received only non-exclusive licenses to contributions to enforce restrictions contained in the outbound open source license. So, for example, the Free Software Foundation, the organization that created the GPL, will frequently request assignments of copyright so that it can more readily take legal action against parties that do not abide by the restrictions on use of GPL software in ways that limit user freedom.

What are some primary concerns about Contributor Agreements?

As the open source software movement has matured, contributor agreements have become increasingly important as a tool to permanently document the contributor's intent that their contributions be shared with the community. The primary concern with contributor agreements is that they be obtained promptly from all contributors. However, as these agreements have grown in sophistication, they have also begun to address other important issues.

One area in which contribution agreements have evolved is that most modern forms of contribution agreement will expressly deal with patent rights, and will state that the contributor is granting a license with respect to any patents held by the contributor that may be implemented by the software. The purpose of the patent license is to provide assurances that the contributor has not submitted software that infringes his own patent. The inclusion of patent licenses raises additional questions, however. What is the breadth of the patent license? Does it cover the project as it exists at a certain point in time, or would it also cover future versions or functionality of the software? Should a patent holder be allowed to assert their patent defensively against someone who has sued them for patent infringement?

Many contribution agreements that include a patent license state that the patent license will terminate if you ever sue a contributor for patent infringement. The idea is that it is only fair to allow a contributor to use his or her patent defensively. Some contribution agreements are also being updated to expressly deal with so-called “moral rights,” the rights of an artist or author, separate from any copyright, to protect their work from misattribution or editing that would destroy the integrity of their work.

There was little disagreement on these issues, except for the terms of the patent license that could be offered by institutional contributors, which consumed most of the Summit time. The key issues were (1) whether institutions such as research universities can give a patent license to all the patents they may have some sort of interest in, or whether the patent license should only apply to patents where the contributor is also the inventor with respect to that patent, and (2) the need to clarify that any license is subject to the terms of any funding agreement that accompanies the governmental patents held by institutions involved in the open source software project or private grant. One output of the Summit was CCLA 1.1, a new form of contribution agreement that attempts to address these concerns. The new contribution agreement will be implemented by both Kuali and Sakai.

See also [Patents and Third-Party Patent Licenses](#)

What should I know about “outbound” licenses?

When software is released, it comes subject to an outbound license that specifies the terms and conditions under which the code may be used. Outbound licenses have traditionally been chosen by institutions based on institutional familiarity and sometimes philosophy. Not all licenses are the same, especially when it comes to the terms on which redistribution of the software is permitted.

For example, the GPL generally obligated people who wish to use software that is subject to the GPL in a project of their own to distribute the software they create only under the same GPL terms. This effectively prevents the software from being incorporated into closed, proprietary software. Other licenses, such as the BSD or Apache, are more permissive, enabling open source software projects to include the software in larger works that are distributed under a different license -- including re-distribution with closed code. Indeed, it is generally believed that there is BSD code in many proprietary software packages.

The Summit group generally preferred an Apache-type of license that does not require that a user who makes modifications release his modifications under the same form of license. However, the group wanted a license that would meet the unique needs of the higher

education community, particularly in regard to patents. The outcome of the Summit was the ECL 2.0.

See also [Developer Issues](#)

What should I know about “inbound” licenses?

There is a tremendous amount of open source software available that provides standard functions (calendars, communications, authentication, databases, etc.) in the form of software “libraries.” Using this code allows the project to avoid re-inventing the wheel by taking advantage of the huge amount of open source code available to the community.

The authors of this third-party software distribute it under a free or open source license of their choice.

It is important to review the license under which third- party software is made available, however, in order to make sure it is compatible with the project’s intended form of outbound license. For example, if an inbound license says “all redistributions must be under the GPL,” then the project’s outbound license must be the GPL. If it is not, the library cannot be included in the project’s release.

What if the licenses on existing components are incompatible?

If software libraries are subject to a form of inbound license that is incompatible with the form of outbound license chosen by the project, either the software libraries may not be used, resulting in additional work to re-create the necessary functionality, or permission must be sought from the author.

License incompatibility can result in unnecessary costs and delay, making it essential to review the inbound license for compatibility with the intended outbound license at a very early stage.

The consensus of the Summit group was that the licensing process should be streamlined as much as possible to avoid license incompatibility and make it as easy as possible to determine if a license is compatible with the project’s intended outbound license. A major point of discussion was whether to recommend a small number of “standard” recognized open source licenses such as Apache or BSD, whose compatibilities are well-known, and avoid the use of third-party software that is subject to a less-standard license that would require additional review and analysis.

What are the best practices to streamline the incorporation of software in a way that ensures license compatibility?

Both Sakai and Kualu projects have built-in quality processes.

Sakai has a “pre-emptive” license review process for inbound code. The code undergoes a license review before incorporating it into the project software and it is checked against lists of acceptable forms of inbound license. Specific individuals have been designated with responsibility for quality control to assure license compliance.

Kuali has taken the additional step of using proprietary software tools, which are available from intellectual property management companies such as Palimida and Black Duck, to scan its code as a base way of verifying that their efforts had identified all of the third-party software used in the project. This work can also be outsourced to service providers such as law firms to conduct an assessment to ensure the “cleanliness” of the code, but the costs of outsourcing this work can be prohibitive for an open source initiative. While asking developers to pay attention to inbound license before relying on third-party software requires a little additional effort from the developers, and formal code review processes can be time-consuming or expensive, they are well worth it in order to minimize risk and protect a project’s reputation. The adoption of strong quality control processes across various initiatives will also lead to efficiencies as these initiatives will be able to share code between projects more readily.

What license should be used for higher education projects?

ECL 2.0 has been created to meet the needs of higher education and broad adoption will ensure consistency.

What contributor agreements should be used for higher education projects?

Kuali and Sakai use a form of contribution agreement for individuals that is identical in all substantive respects to the form of contribution agreement used by Apache. A new form of contribution agreement based on the Apache model that will be used for institutions that have staff contributing to projects was created at the Summit.

6.0 PATENTS AND THIRD-PARTY PATENT LICENSES

Discussion of how patents should be addressed in connection with collaborative open source projects consumed about half of the Summit time. The key issue was whether institutions can give a wide patent grant that would cover all the patents they have an interest in - including patents that may have arisen out of the work of individuals who are not contributors to the open source project, or that may have arisen out of work funded by third parties.

The belief among some institutions was that the commonly used Apache form of contribution agreement required a patent license that was too broad, because it did not take into account the possibility that a

contributor institution would have previous commitments to third parties to whom the patent had already been licensed, or who may have funded or participated in the relevant research. For some of the larger research universities, simply determining the relevant patents and agreements can be a daunting logistical task.

Two outputs of the Summit addressed these concerns: the ECL 2.0 outbound license and the new form of institutional contribution agreement. These revisions were designed to accommodate concerns about the reach of the patent license provisions in the contributor agreement. The patent license provision was modified so that no license would be granted to patents developed by anyone other than the author of the contribution, and also to recognize the possibility that there may be funding agreements or other prior commitments that limit the institution's flexibility to grant a license.

While these licenses represent progress, they also reflect some policy decisions by participating institutions that bear long-term thought. For example, a license to patents that arise only out of the work of contributors to the project does not cover patents that arise out of other work at the university, reflecting a choice to protect the ability of individual inventors at the university, and the ability of the university itself, to benefit from the commercialization of the patent, where licensing these patents in connection with community projects may be beneficial to the community as a whole.

7.0 DEVELOPER ISSUES

Individual developers who may become contributors to an open source project often have their own concerns that differ from institutional concerns. The status of individual developers differs based on their working arrangement with an institution. At many institutions, the software developed by IT staff and contractors is deemed "work for hire" and all rights are held by the university. Faculty members, however, will typically have more rights in the intellectual property they create, such as royalty-sharing policies.

Part of the Summit discussion was related to how to balance faculty rights with the desire of the universities to support the efforts of open source in higher education. The new patent language in the institutional collaboration agreement is intended to reflect the fact that at many universities, the royalty-sharing policy does not contemplate non-commercial uses of the patent, and therefore leaves uncertainty as to the universities' ability to permit such non-commercial uses.

What is my responsibility as a developer related to contributor agreements?

There are two issues, depending on what you are contributing to the project. If the software is your original creation, you must read and sign a contribution agreement. If you are including someone else's software, you should be aware of and document the type of inbound license under which source code is available. What does the license require? What does it allow you to do with the code you are using? Is it compatible with the project license? Even if you identify a compatibility problem, it may be possible to contact the original author and request an additional - compatible - license.

See also [What are contributor agreements and “inbound” and “outbound” licenses?](#), [Who signs and what terms typically govern Contributor Agreements?](#) , and [What are some primary concerns about Contributor Agreements?](#)

When I am using open source code, what is my responsibility for determining whether the intended use complies with its license?

Projects assess code in different ways. Sometimes individual developers are responsible for checking the licenses governing code they wish to use in the project. This is common for non-institutional open source projects. Other projects have more established processes. Check with your project manager to know how your project assesses code, but at a minimum it is important to realize that you are responsible for identifying any third party code you have incorporated into your contribution, and it is also important to recognize the possibility for wasted development time if attention is not paid to compatibility issues early.

See also [What are the best practices to streamline the incorporation of software in a way that ensures license compatibility?](#)

What is my responsibility as a developer related to outbound agreements?

If you are participating in an open source project, whether directly as a project developer or indirectly as contributor, it is essential to be aware of the type of agreement under which the source code is being released - the outbound license. This sets limits on which code can be incorporated in the development.

See also [What should I know about “outbound” licenses?](#)

What if my institution's policy conflicts with a license?

If institutional policy prevents code from being contributed, you should work with the open source project to see if license adjustments are possible that will be mutually acceptable to both your institution and the project.

The changes in the ECL were written to address most major institutional issues and to make it easier to collaborate on open source software

projects, in hopes that institutions should have no conflicts with participation.

What is the risk to individual code contributors?

There is no risk of liability for copyright infringement if individuals contribute only software they have written themselves, and to which they hold the copyright, and if they clearly disclose when they are contributing software that others have written. The contribution agreement provides an opportunity for the project to ask contributors to represent that they are contributing your own work, and have identified the source of any software that is not their own work. It also includes an express disclaimer of liability if the software does not work as intended.

Who owns a collaborative software project?

The copyright to software written by university faculty or staff may reside with either the individual author of the software or the institution, depending on the terms of their employment agreement and the policies of the institution. At many institutions, the default rule is that faculty will own anything they author, while software developed by IT staff will be owned by the institution.

See also [What are contributor agreements and “inbound” and “outbound” licenses?](#) and [Who signs and what terms typically govern Contributor Agreements?](#)

What if the software is modified?

Whether modifications are permitted will be governed by the terms of the inbound license under which the software is used. In some cases, the ability to make modifications is subject to conditions, for example, that the changes made to the code be clearly documented.

See also [What should I know about “outbound” licenses?](#)

Can the software code that I have contributed be used commercially?

The terms of the outbound license determine whether any commercial use must remain open source or is allowed to be incorporated into a closed, proprietary system. The ECL 2.0 allows any use.

See also [What should I know about “outbound” licenses?](#)

8.0 PATENTS AND COPYRIGHTS OUTSIDE OF THE UNITED STATES

Many developers and institutions involved in open source projects are from outside of the United States. This means there are software contributions both from and to the US, as Americans contribute to European or Japanese or

Australian projects and developers in these countries send code to US projects. Although the most commonly used open source licenses were drafted within the United States, there is now a worldwide community of developers and contributors, introducing the potential for greater complexity and requiring licenses and contributor agreements to reflect this new reality.

How does copyright law differ outside of the United States?

The main legal basis for protecting software programs is substantially harmonized on a global basis and copyright protection arises automatically: no registration requirement for protection, no requirement that a sample of the work be deposited in a central repository, no fees, etc. A series of international treaties signed by most of the nations of the world ensure that works created in one country will be protected by laws in another. Countries have basically agreed to provide foreign authors the same degree of protection as they do to local authors, without requiring any registration or other procedure. Therefore, code is protected wherever it goes, and rights holders have equivalent rights to protect their code “here” or “there.” Although there are minor differences, (i.e., what is called a “distribution” in the US may be considered a “communication to the public” in Europe), in the end, permission for the way the software will be used and shared must be granted by the licenses. It is important to draft licenses and contributor agreements that are as clear and “generic” as possible, so that a judge interpreting and enforcing them in another jurisdiction has less difficulty determining what rights are intended to be granted.

How does patent law differ outside of the United States?

There are substantial differences between the US and other countries (or areas, such as Europe) with regard to patent protection. Patents create difficulty in the open source context, as the community is generally opposed to applying patents to software. While in the copyright context, numerous international agreements provide relatively worldwide protection, in the patent field, what can be protected and the degree of protection is much more territorial: national (or regional) patent offices grant patent protection for a specific nation or region. European law (based on the Munich “European Patent Convention”) is unclear about whether software processes or products may have patents applied to them. Textually, “computer programs” are excluded from patentability BUT this exclusion is subject to a proviso that says only “software as such” is excluded. This has allowed the European Patent Office to issue software patents that have “a technical effect beyond interaction with the computer.” As a result, several software companies have been able to apply for, and obtain, software patents. However there is a trend against this, both at the European level, where it appears that the EC may have given up trying to gain patent protection for software) and the national level (e.g., England is reviewing its patent law related to software, and a string of recent cases has questioned the concept, while not necessarily being determinate about it). This means that patents that are

recognized in the US may not be granted in Europe, and if they are, the courts may be more likely to strike them down.

With that said, it is important to consider the patent situation in the US and other major markets even for projects that do not originate in the US because open source software is typically distributed worldwide via the Internet.

How are warranties and indemnities different outside of the United States?

Another area of difference, though again not major, is related to warranties and indemnities – or, in the case of open source software, disclaimers and limitations on warranties and liability. The European legal framework, at least, imposes certain warranties, (e.g., title) on software distributors, that may, arguably, be excluded in the US. This is because mandatory law protects weaker parties (consumers, in particular) from unfair unilateral licensing practices. While these differences may not be so great in a professional context, it is important to consider this when dealing with consumers – especially when a project distributes code around the world. Again, this may influence the project license (as these exclusions are contained here) and management of responsibilities within projects, which is one of the reasons for setting up foundations and other institutions to manage the code.

Which law applies?

Some licenses leave the decision of which law applies when interpreting and enforcing the contract up to the underlying “conflict of laws” law (or “private international law,” as it is called on the European Continent), which has evolved several principles to help courts decide if they are competent to hear a case and, if so, which law to apply. If a developer does not want to rely on these principles, a license may specify that the law of the software provider will apply (and its courts have jurisdiction) while others specify a specific jurisdiction. Even so, there is enough uniformity in copyright law generally for open source projects to thrive with many participants from many jurisdictions, all working under standard US-style contribution agreements and licenses.

What if international developers join in a collaboration with US foundations on open source projects?

When international developers work with US open source initiatives, US law treats them as rights holders whose rights will be protected under copyright law. The risk of this code infringing third party rights (patent or copyright) is as high or low as that of any other contribution. As rights holders, international developers will be required to enter into a contribution agreement with the open source project – and agree to any patent license – so that the project will have the freedom to redistribute the code. One of the goals in drafting contribution agreements is to draft them in simple enough terms that they will be easily interpreted and considered valid in any country that an author may reside in.

Even though contributions will generally be governed in the same way as US contributions, open source projects should still carefully consider the international legal dimensions of any collaborative project.

APPENDIX A. SUMMIT PARTICIPANTS

Co-Chairs

- Daniel Greenstein, Associate Vice Provost, Scholarly Information, University of California Office of the President
- Brad Wheeler, Chief Information Officer, Indiana University

Organization	Representative(s)
Apache Software Foundation	Cliff Schmidt, Director and Vice President, Legal Affairs
Australian National University	Robin Stanton, Chief Information Officer
Cambridge University (UK)	John Norman, Director - Centre for Applied Research in Educational Technologies
Cornell University	Sandy Payette, Co-Director, Fedora Project and Researcher, Cornell Information Science
DSpace	McKenzie Smith, Associate Director for Technology, MIT Libraries
IBM	David Shields, Program Management, Linux Technology Center
Ithaka	<ul style="list-style-type: none"> • D. Barnaby Gibson, General Counsel, Treasurer and Secretary • Matthew Rascoff, Analyst, Strategic Services
Indiana University	<ul style="list-style-type: none"> • Beth Cate, Associate University Counsel • Jack Pincus, Vice President of Technology Transfer at the Indiana University Research and Technology Corporation
Joint Information Systems Committee (JISC)	<ul style="list-style-type: none"> • Ralph Weedon, Director , JISC Legal Information Service, University of Strathclyde (UK) • Naomi Korn, JISC IPR consultant
Kuali Foundation	Barry Walsh, Senior Director, eBusiness Services, Indiana University
LegisTICs, Barcelona, Spain	Malcolm Bain, Partner
The Andrew W. Mellon Foundation	<ul style="list-style-type: none"> • Jacqueline D. Ewenstein, Assistant General Counsel • Ira Fuchs, Vice President of Research in Information Technology • Don Waters, Program Officer, Scholarly Communications
University of Michigan	Paul Courant, University Librarian and Dean of Libraries
MIT	Karin Rivard, Assistant Director and Counsel, Technology Licensing Office
OSS Watch	Randy Metcalfe, Manager, OSS Watch, University of Oxford (UK)
Pennsylvania State University	Jim Leous, Manager UNIX Systems and Technical Solutions Group
the rSmart Group	Chris Coppola, rSmart President and director, Sakai Foundation and Kuali projects

Open Source Summit Educational Materials 1.0

Sakai Foundation

Joseph Hardin, Director of the Collaborative Technologies Laboratory,
University of Michigan

University of British Columbia
(CAN)

Randy Smith, Technology Transfer Manager

Univeristy of California Office of the
President

- Stephen Benedict, Director IT Strategic Sourcing, Information Resources and Communications
- Charles Drucker, Technology Transfer Officer
- Mary MacDonald, Counsel

The Summit was sponsored by the Andrew W. Mellon Foundation.

Tina Howard produced this document.