OSBR.CA

The Open Source Business Resource

DECEMBER 2007

DECEMBER 2007

PUBLISHER:
The Open Source Business Resource is a monthly publication of the Talent First Network. Archives are available at the website: http://www.osbr.ca

EDITOR:
Dru Lavigne
dru@osbr.ca

ISSN:
1913-6102

ADVERTISING:
Rowland Few
rowland@osbr.ca

GRAPHICS:
Ryan May

As 2007 draws to a close, three emerging trends are gaining momentum. The first is that companies are releasing formerly proprietary code under an open source license. The second is that open source companies are being acquired or are issuing public offerings. The third trend is that very large number of citizens increasingly uses the Internet to oppose politicians and law makers who threaten, sometimes unwittingly, the fundamental principles of open source development.

These three trends tie into this month's editorial theme: Clean intellectual property or clean IP. In a nutshell, clean IP is about reducing license incompatibilities and non-compliance with licensing terms. Clean IP significantly affects the value of the code released as open source and the value of a company that develops and markets software.

One takeaway from this issue is that those companies with clean intellectual property (IP) stand to gain significant competitive advantages. In the first article, Doug Levin, the CEO of Black Duck Software, identifies the best practices for managing IP in development environments that use open source components, commercial off-the-shelf components, and proprietary components. Then, in the second article, Mahshad Koohgoli and Richard Mayer, Protecode's CEO and Vice President of Marketing respectively, examine the various methods of ensuring software IP cleanliness.

In the third article, Andy Kaplan-Myrth from the University of Ottawa discusses the importance of consultation process for the upcoming legislation on Canadian copyright. He is followed by two academics, G.R. Gangadharan (University of Trento) and Michael Weiss (Carleton University), introduce rights expression languages. These languages express rights, fees and conditions as machine actionable functions.

Eric Smith, a lawyer with the firm Fraser Milner Casgrain, explains the changes introduced in version 3 of the GPL. In the last article of this issue, Russell McOrmond, an Internet and F/LOSS consultant, provides a perspective on the copyright bill expected to be introduced in early 2008. Many Canadians have expressed concerns about this bill.

In addition to the six articles, this OSBR issue includes a call for proposals from the Talent First Network and a letter from David Fewer inviting our readers to join the Canadian Software Innovation Association, a coalition concerned about the negative consequences of importing US-style digital copyright legislation to Canada.

2008 will see some exciting changes for the OSBR. The first will be a move to a new open source publishing system. OSBR readers, authors and reviewers will benefit from the functionality of the new system.

Happy holidays.

As always, we look forward to reading and publishing reader feedback.

Dru Lavigne,

Editor-in-Chief

dru@osbr.ca

Dru Lavigne is a technical writer and IT consultant who has been active with open source communities since the mid-1990s. She writes regularly for O'Reilly and DNSStuff.com and is author of the books BSD Hacks and The Best of FreeBSD Basics.

"Software assembly, whereby developers re-use already created code components, is dramatically changing how applications are built. The efficiencies are realized if development teams establish processes to manage how code is assembled, and to assure that the use of the code components is in compliance with the various licenses governing the code."
Doug Levin, CEO of Black Duck Software

Over the past ten years, the Internet and open source software (OSS) have enabled developers to fundamentally change the way they produce software. Increasingly, distributed teams are collaborating to assemble software from reusable components and their own proprietary code rather than building applications entirely from scratch.

The component-based development model is fundamentally changing the software industry. It enables organizations that develop software, either for commercial sale or for in-house use, to accelerate project timelines, improve software quality, and reduce development costs. If not managed properly, the complexity inherent in this new world of mixed-IP (Intellectual Property) can pose business and technical risks to an organization.

This paper draws on the experiences of the Black Duck Software team (http://www.blackducksoftware.com), our customers, and other industry experts to propose new approaches to managing IP in this new world. It describes a set of best practices that companies can use to avoid the risks and gain the benefits of the component-based approach to software development.

Component-based Development

Development organizations have long understood the virtues of building new applications by re-using components already built and tested. Indeed, the evolution and rapid adoption of component-based architectures has been driven in part by their effectiveness in promoting economically-significant re-use. It is therefore appropriate to consider components as IP assets, optimize their usage, and protect their integrity. This applies equally to all components, whether they are internally developed or developed externally by a third party.

A development team intent on exploiting an externally developed commercial component does not typically purchase ownership of that component. Instead, they acquire a license to use that component in a specified manner--perhaps for only a certain number of developers working on a particular project, or with a specified royalty paid for each instance of a shipped product that includes the component. Thus, business judgment is required to ensure that the cost of licensing a commercial component is more than offset by the benefits.

Included in the cost analysis must be the effort required to ensure compliance with the license; for example, limiting the component's usage to a specific project, or tracking shipments in order to accurately calculate royalty payments.

Open Source Opportunity

Over the past five years, OSS has risen to prominence, dramatically increasing the opportunity to re-use existing software. Developers can readily locate potentially useful components from a wide array of re-usable software components.

Re-use can take many forms, including bundling independent components, integrating with or using libraries, and incorporating source code or source code fragments. In some cases these components can be modified as required to improve functionality, quality, performance, or footprint. As with commercial components, the ownership of externally developed OSS components and fragments remains with their authors. While most of these authors allow the commercial use of their software without initial payments or royalties, many have chosen to impose other constraints, such as:

- Attribution
- Usage reporting
- Publication of modifications and improvements
- License replication
- Resulting software must be open source

Such constraints are imposed by means of licenses. Examples of open source licenses include the:

- Apache Software License (ASL)
- Common Public License (CPL)
- GNU General Public License (GPL)
- Mozilla Public License (MPL)
- New BSD License

A more complete listing of open source licenses is provided at http://www.opensource.org/licenses.

Open Source License Compliance

Development teams that incorporate OSS components or fragments of OSS components in their projects must comply with the terms of the licenses associated with those components. This can be challenging for several reasons including the following:

- There is wide variation in the obligations imposed by open source licenses, ranging from the BSD license (which has few obligations) to the GPL license (which has many)
- Some open source licenses are legally complex, introducing constraints whose business implications may not be obvious to a developer choosing to re-use a component
- The licenses of some commercial and open source components are mutually incompatible
- The origin of a particular source code fragment may be difficult to determine, effectively obscuring its license obligations
- Discovering the need to comply with a license late in a project's lifecycle can produce disagreeable tradeoffs, such as publishing all of the project's source code v.s. increasing time-to-market by months while a component is replaced

In addition to the legal obligations imposed by licenses, developers who incorporate OSS components into an organization's projects may either be obligated by the terms of the license or feel a moral obligation to give something back to the community. The resulting actions may result in the inadvertent dilution or loss of the organization's IP.

Management Alternatives

Organizations can react to the challenges of OSS licenses in one of three ways. Some organizations turn a blind eye, ignoring the issue until a catalyzing event or crisis occurs. But the resulting misfortunes--major code rewrites, embarrassing negative publicity, delayed sales, failed acquisitions--make this an increasingly untenable approach. This is especially true in the new environment of increased business transparency, executive responsibility, and potential shareholder lawsuits.

Other organizations take the Draconian approach of banning all OSS re-use. This strategy is flawed because it:

- Is difficult to enforce
- Decreases productivity and agility compared to organizations that successfully re-use externally developed components
- De-motivates development teams by requiring them to apply scarce resources to develop components from scratch and test them rather than moving forward

The first two approaches fail to recognize the reality that open source and component reuse are here to stay. The third and recommended alternative is to encourage the re-use of both internally developed and externally developed (commercial and open source) components, while establishing controls at critical points in the project lifecycle, for example:

- When components are first added to a project
- When internally developed components are created or modified
- At every build
- At each phase transition in the development process
- When considering the contribution of a component to an open source project or the transfer of its ownership to another party
- Before acquiring a significant ownership interest in a software component

It is important to note that identifying problematic licensing issues early in the development cycle is tantamount to detecting serious software defects. The earlier a problem is detected, the less expensive it is to fix. While IP controls late in the development cycle, during testing or release assembly, are better than none, the earlier they happen in the lifecycle, the better.

Preparing for IP Management

To adopt best practices, several tasks should be considered by the individual or teams responsible for development and licensing. An organization intent on managing its software IP should identify the responsible business, legal, and technical individuals who will be involved in the process. The organization also should designate them as authorizers for each active project, and commission them as a group to oversee and manage the planning, implementation, and ongoing management of the process.

A good first step for the team is to define the boundary between internally developed and externally developed components. For example, a small organization that prefers taking a conservative approach may deem all of the code developed within its walls to be internal. Conversely, that company would view as external all software brought in from any outside source.

Another organization might extend its view of internal software to include licensed proprietary software and software developed by its contractor and outsource partners. On the other hand, a department of a large corporation may want to consider its department internal and consider everyone else, including other departments in the same company to be external. Business judgment must be used to determine where the boundary should lie.

Another key preparation task is for the team to identify the development process phase transitions at which component reuse reviews will be conducted. The team should define criteria for designating an internally developed component as sensitive, for example due to trade secrets or patents, and develop and maintain a list of these sensitive items. The organization also should consider establishing and maintaining lists of:

- Licenses that are prohibited by the organization
- Externally developed components that, based on previous reviews, are approved for use in projects, and the situations in which use is approved
- Internally developed components that, based on previous reviews, are approved for contribution to open source projects or disposition to third parties

Once these lists have been created, the organization can use them to conduct an initial assessment of its existing code base(s). In this important preparatory step, the organization identifies and establishes the baseline pedigree, licenses, and components in use. As with any process improvement, an acceptable alternative approach involves introducing these steps incrementally over time.

Best Practices for IP Management

Whenever a development team considers adding new features or refining existing functionality in a project, it should explicitly seek internally developed and externally developed components that could accelerate delivery. The team should establish criteria for selecting and procuring these components. As would be expected, any component under consideration that fails to meet functionality, performance, reliability, maturity, or risk requirements should be eliminated.

The team should also eliminate any externally developed component whose license is on the prohibited license list, whose license obligations are financially or legally incompatible with the project's business objectives, or which uses other external components or fragments whose licenses are similarly unacceptable. For example, an organization developing a product that will be delivered under a proprietary license needs to be certain that any open source or proprietary licensed code that is incorporated can be safely included without causing an irreconcilable conflict between licenses. Any components that pass this initial test should be subjected to a make-buy analysis to determine whether or not its acquisition makes sense from a business perspective.

To protect the organization's critical IP, the creation and modification of all internally developed components should be tracked by recording a timestamp, the names of each author, and the applicable objectives and constraints. If a newly-created or modified component is suspected to be sensitive, the project's legal, business, and technical reviewers should be convened. If these reviewers deem the component to be sensitive, they should add it to the organization's list of sensitive internal components.

By assessing sensitivity and license obligations at the point where a component is first being considered for re-use, decisions can be based on verifiable facts, eliminating last-minute surprises, guesswork, compromises, and risk-taking. This dramatically reduces the risk of schedule slippage, cost overruns, and damage to the organization's reputation. It also helps prevent the inappropriate re-use of critical IP.

For each component that a project's development team proposes to use within a project, the team should understand:

- The intended use and rationale for inclusion
- The component's sensitivity
- How the code will be incorporated

How the team deals with the component will depend, in part, on whether plans call for that component to be used temporarily or permanently. For example, the intent may be to use that component for a limited amount of time only to speed up prototyping or to advance the early phases of the development cycle, but not be intended to be made a permanent part of the code base.

Another determination the team should make is whether a component will be used only as is, or if modifications will be allowed, and if so, under which approvals. Development teams should describe the method of joining that will be used to incorporate the component into the project. This is an effective step because different types of joining can create different licensing obligations.

To achieve greater control over component re-use, teams should also take the following actions:

- Determine whether the component has been previously approved for the proposed form of use
- Declare the component's version and understand its license obligations as well as those of any externally developed components or fragments it contains or depends on
- Understand all potential incompatibilities between the component's license obligations and the license obligations of other externally developed components included in this project
- Present the above information to the project's legal, technical, and business authorizers and request approval to use the component as described
- If the component is internal and sensitive, the list that covers these items should be updated to note that component's inclusion in this project
- If the component is externally developed, its metadata and approval details should be recorded in the list of approved external components

Inspecting the code base on a regular basis decreases the likelihood that unexpected components will be introduced without being noticed. Therefore, at the creation of each project build or at release assembly, the development team should verify that:

- No unapproved sensitive internally or externally developed components or fragments have been added to the project
- No unapproved changes have been made to sensitive internally developed components

- No changes have been made to externally developed components whose form of use precludes changes, or requires that all changes be approved

Any inappropriate additions or changes discovered during verification should be immediately addressed, either by obtaining approval from the project's legal, technical, and business reviewers, or by backing out the offending modification. The root cause of any component misuse should be identified and corrected to ensure no subsequent regression. By promptly and diligently assessing every build and release, the development team will be able to detect errors when they are least expensive to correct.

At the completion of each build or release, the key metadata for all externally developed components should be recorded in the associated bill-of-materials. This enables demonstrable compliance with license obligations, and eliminates any uncertainty caused by changes between project releases by providing a clear audit trail. As a project completes a major development process phase, its legal, technical, and business reviewers should do the following:

- Verify that no unapproved sensitive internal or external components or fragments are used in the project

- Verify that no unapproved changes were made to sensitive internal components, and that no unapproved or precluded changes were made to external components

- Review the license obligations of all external components used in the project, and ensure compliance with these obligations

These phase reviews backstop the development team, and keep the legal, technical, and business reviewers engaged in the management of software re-use. They also verify that changes in the project's objectives have not created legal, technical, or financial inconsistencies with the licenses of components used in the project.

The rationale for contributing components to an open source project is beyond the scope of this report, as are the considerations involved in transferring ownership to a third party or creating a new open source project. However, if a contribution or transfer of a candidate component or fragment is deemed appropriate, the project's legal and business reviewers should:

- Determine whether the candidate component's sensitivity is an impediment to contribution or transfer

- Verify the right to contribute or transfer every externally developed component or fragment contained within the candidate

This helps to ensure that the organization does not inadvertently contribute code that shouldn't be added because of its sensitivity or because the organization is not entitled to supply it. If an organization is considering an acquisition that would include a significant interest in one or more software components, the designated set of legal, technical, and business reviewers should be charged with the following:

- Identifying all included components not owned by the supplier or target

- Assessing their license obligations with respect to the acquirer's compliance, business objectives, and legal policies

- Assessing the impact of any required rework or change on cost, revenue, and quality

Note that this best practice applies to a variety of situations in which financial investments are involved. Such situations include: company mergers and acquisitions, product acquisitions, joint venture formations, and venture capital investments.

Conclusion

By integrating these practices in to development processes, organizations will have far greater assurances of compliance with all relevant license obligations and far more effective protection of software IP. Adopting these practices will enable companies to be more aggressive in their use of the software assembly approach. That, in turn, will enable those companies to more quickly gain the benefits and competitive advantage this new development approach promises including accelerated project timelines, improved software quality, and reduced development costs.

Recommended Resource

Best Practices for Managing Software Intellectual Property
http://www.blackducksoftware.com/media/_wp/ManagingSoftwareIP_BP.pdf

Doug Levin is president and CEO of Black Duck Software. Prior to founding Black Duck in 2002, Levin served as CEO of MessageMachines (acquired by NMS Communications in 2002) and X-Collaboration Software Corporation (acquired by Progress Software in 2000). From 1995 to 1999, he worked as an interim executive or consultant to a range of software companies, including CMGI Direct, IBM/Lotus Development Corporation, Oracle Software Corporation, Solbright Software, Mosaic Telecommunications, Bright Tiger Technologies and Best!Software. From 1987 to 1995, Levin held various senior management positions with Microsoft Corporation, including heading up worldwide licensing for corporate purchases of non-OEM Microsoft software products. He is a graduate of the University of North Carolina at Chapel Hill and holds a certificate in international economics from the College d'Europe in Bruges, Belgium.

"By June 2006, the project has hit the magic "100 cases finished" mark, at an exciting equal "100% legal success" mark. Every GPL infringement that we started to enforce was resolved in a legal success, either in-court or out of court."
http://www.gpl-violations.org

At many points in the life of a software enterprise, determination of intellectual property (IP) cleanliness becomes critical. The value of an enterprise that develops and sells software may depend on how clean the software is from the IP perspective. This article examines various methods of ensuring software IP cleanliness and discusses some of the benefits and shortcomings of current solutions.

Source of the Problem

In any software-producing organization, software projects conceptually comprise iteratively decomposing a project into smaller and more manageable sub-projects until that point where individual sub-projects can be assigned to individuals or groups. In such a scenario, an individual software developer assigned to a software sub-project has a combination of options for sourcing the necessary software, including: (i) obtaining software from their organization's code repository, (ii) obtaining software from one or more open source repositories, (iii) obtaining software through purchase, and (iv) obtaining software through the creative process of writing it.

Depending upon the size and complexity of a software project, this scenario may be repeated many times. Also, when a sub-project is outsourced or otherwise assigned to a separate organization such as in a collaborative project, the outsourcing team goes through a similar scenario.

In sum, the software employed by any typical software-producing organization may be derived from such sources as: (i) the organization's code-base; (ii) open source repositories; (iii) commercial vendors; and (iv) in-house development.

With this scenario in mind, IP integrity of the final product is very much a function of: i) individual pieces and ii) the IP practices that were defined, monitored and enforced during the development process.

We observe that the IP integrity of the software produced can be compromised in ways that include:

- The organization's code repository may have impure artifacts that are introduced into the product
- If used, the outsourcer's repository or the collaborative organization's repository may have impure artifacts that are submitted
- Open source components do not satisfy the open source policy of the organization, assuming that the organization has an open source policy, and it has been appropriately communicated
- Open source components introduced by outsourcers or collaborators may not satisfy the open source policy or may be improperly checked and verified against existing policies
- The license governing the use of certain commercial code may be improperly licensed for the application, geographic market, or the mode of deployment

In addition, a developer may make a contribution that is copyright by another firm.

For example, a recent survey indicates that about 70% of developers carry code from one company to another (http://www.itfacts.biz/index.php?id=P1134). Anecdotal experience suggests that the real percentage may be higher.

The situation gets more complex if we consider that code repositories can become contaminated because of "generational tainting" properties of licenses such as the GPL, or more generally, the pedigree of open source code may be unknown or difficult to categorically establish.

In this article, we do not address intentional code contamination and instead focus on unintentional contamination as we believe that most code contaminations are unintentional. For any software project of sufficient size it is generally difficult to understand exactly what is in the software by the time the software is collected, integrated, tested and released. It may be very costly and time consuming to perform the necessary due diligence to identify what problems may exist.

An immediate conclusion we can draw from the preceding is the importance of employing safe software development practices. That is, we advocate a preventive approach aided by policies, education and tools.

Who is Affected

Understanding the IP pedigree of software is important and is becoming an increasingly common requirement in many enterprises that create and/or use software. In what follows, we first describe the software food chain, and then examine how it can impact the players.

The software industry chain may be described in simple terms.

The chain consists of increasingly larger and more complex organizations that "consume" software by bringing in software from other (typically smaller) firm(s), combining the software with their own "value-added" functionality, and passing the result on to the next (typically larger) firm in the chain.

To show the scope of the firms that can be affected by contamination, let's use an example. The software chain in the cellular phone industry consists of:

- Small developers and independent software vendors that contribute hardware drivers or protocol stacks to the chip makers
- Chip makers that add their own content and pass it on to the cell phone vendor
- The cell phone vendor adds applications and graphical user interfaces, again obtained from other players and internal development teams, and passes these on to the operator
- The operator may add its own customized content such as splash screens or operator-specific applications and pass these on to the end user

In this example, there could be twenty or more players involved. An IP contamination anywhere in the food chain would affect many players.

Attention on IP purity is heightened generally when there is a transaction involved. The nature of the transaction could be:

- Investment in an organization that creates or consumes software
- An M&A (merger and acquisition) activity involving a player in the above food chain

- Public Offering by a player in the chain

- A software transfer from one player to the next in the chain, such as a contracting or software licensing event

Such transactions contribute to the creation of a whole industry centered on verifying software cleanliness through due diligence. Another contributor is the clauses on representations and warranties or IP indemnity in legal documents supporting such transactions. It is increasingly common to encounter IP lawyers and Venture Capitalists (VCs) and be regaled by stories of transactions that have been delayed or lost (i) due to the time that it takes to verify cleanliness, or (ii) due to ambiguities around IP ownership. All of which advocates the need for adopting and deploying safe software development practices.

IP Contamination is Prevalent

The scope of IP contamination is expanding, and is better understood when we examine its contributing factors and the momentum behind them.

The use of open source software (OSS) is increasing dramatically, shifting the models and metrics behind software development. Software re-use, code visibility, efficient development intervals, costs, and enhanced functionalities are some of the positive attributes driving the increasing use of open source. However, very specific licenses regulate the use of OSS and the terms of many of these licenses differ and not uncommonly require expert interpretation.

Aside from OSS, other growth areas for contamination include:

- Designer previous-life contamination, which we described earlier, is common

- Outsourcing, on-shore or off-shore, is expanding, with the additional danger of cross-project contamination

- E-bidding for software as in http://www.elance.com is growing

- Collaborative development is becoming common with universities, governments and industries as players

In sum, we reiterate our belief that most contamination is unintentional, and happens when safe development standards and practices are absent in the software organizations of the food chain.

Current Prevention Methods

We have grouped the general methods of managing IP contamination into two groups: i) corrective methods, which try to detect contamination or IP policy infractions in a piece of software, and ii) preventive methods, that strive to stop unintentional penetration of undesirable code in a project.

We will further divide these methods into manual and automated techniques, and will comment on their suitability and perceived short comings.

Corrective Solutions

Corrective solutions by their very nature require there to be an asset to be analyzed, and therefore are commonly employed as a prefix to the load-build process or as a suffix to the product release process.

The general objectives behind corrective solutions are: (i) to detect possible IP contamination; (ii) to identify the external source from which the IP contamination was derived; (iii) to determine the validity of suspected IP contamination; and (iv) to appropriately respond to the possible IP contamination. IP contamination can take one of two forms: (i) a complete module or file, or (ii) a snippet such as a subroutine or method within a module or file. It is noteworthy, reflecting that IP contamination is seldom malicious, that it is not uncommon for an IP contamination object to have associated comments that clearly identifies the source or copyright owner of the object.

Corrective solutions can be quite sophisticated and there can be value to just performing one or more of the above objectives. For example, it is not always necessary to identify the source from which an IP contamination object was derived. Lexical and grammatical analysis of a file can be employed to detect and flag explanation changes in coding style. Also, detection and identification may be one and the same. A precautionary sweep of all the files used in a load build may be performed to ensure there are no inadvertent and unaccounted for dependencies on commercial or open source licensed software modules.

Corrective solutions typically involve a combination of both manual and automated processes. A comprehensive and formal review may be performed by an internal team of code reviewers and legal council as part of the general development process. This may be an extension to the typical code review that is commonly employed during the development process to find bugs and improve the software.

Alternatively, an external commercial due diligence service may be employed to provide an independent assessment in order, for example, to validate that the software asset being sold rightfully belongs to the seller. In either case, the process involves examining various artifacts such as the software bill-of-materials, software files, and documentation files, determining the sources of various artifacts, and interviewing developers. In short, looking for indicators of external artifacts and upon finding suspects, investigating the IP attributes of these suspects.

The more common commercially available automated tools typically address the identification of external software modules/files together with external source code within modules/files. To accomplish this function, these tools have associated repositories containing software modules/files, actual source code, and the equivalent of code fingerprints known as codeprints. These tools accomplish their objectives by comparing a given artifact against the contents in the repository. Such repositories are commonly assembled by mining the vast collection of generally available OSS as well as contributed commercial software source code, object files, library files, and executables. The contribution of such commercial software creates a win-win for everyone involved as:

- The owners of the commercial software are increasingly confident that their code is being properly used

- The developers of software are increasingly confident that their market offer is not exposed to the risks associated with inadvertently incorporating commercial software

- The tools developer has a more comprehensive and useful market offer

A few examples of the automated mechanisms used to detect and identify external code snippets include:

- Comparison and correlation of the snippet with various snippets of code existing in a repository

- Computing a codeprint for the snippet and comparing this codeprint against a repository of pre-computed codeprints

- Scanning the snippet for keywords such as "license", identification marks such as "written by", and copyright notices

As mentioned, generally a combination of manual and automated methods are employed. Two noteworthy academic projects in this field are: (i) MOSS (Measure Of Software Similarity) in Stanford University (http://theory.stanford.edu/~aiken/moss/), and (ii) JPLAG (https://www.ipd.uni-karlsruhe.de/jplag/) in Karlsruhe University.

Limitations of Corrective Solutions

Although the steps for corrective solutions are relatively straight forward, practically they are very labor intensive, time consuming and expensive. Increasing automation makes this endeavor more feasible, which is contributing to an increasing use of such tools for software development. Which, in turn, is resulting in an increase in the innovation and availability of market offers addressing this area.

However, there are limitations to corrective solutions. Even with automation and aggregation, corrective solutions can not detect external content unless they can identify it. In other words, if an external content is not available in the tools's repository or is not otherwise detectable by being clearly marked or stylistically different, it cannot be detected.

And in particular, it is extremely difficult, if not impossible, to detect proprietary external content since generally it is not available for comparison purposes.

Even if the external content does exist in the database, the comparison process is not always 100% accurate. Moreover, the comparison process is computationally intensive, requiring a tight linkage between the comparison algorithms and the repository. This typically requires co-locating the repository and the code to be examined.

Corrective solutions address the relatively well defined issue of spotting IP contamination. There are, however, a myriad of related and overlapping issues. Notable among these issues is software pedigree, or "who wrote this stuff?". Software pedigree is concerned with determining whether or not the copyright license attached to a file/module or code snippet is properly attached and that the license really does apply. Another issue surrounds determining what constitutes a software derivative as some OSS licenses require all derivatives of a given work to be licensed under the same license as the original.

Possibly the most significant limitation associated with corrective solutions is that they are corrective, occurring after the fact. Resolution of any corrections can impact the project's completion date, transaction closing and sales cycle, and add unanticipated costs to the overall project.

Preventive Solutions

As the saying goes, "An ounce of prevention is worth a pound of cure". As with corrective solutions, preventive solutions may be categorized in terms of manual and automated processes.

Among the more widespread manual preventive solutions is education. Education includes organizations setting policies and rules for acceptable and safe coding practices, as well as the associated communications and training. These include the introduction of guidelines that are well documented, generally available for developer reference, and are integrated into the company's practices.

The success of education in addressing IP contamination is dependent upon a number of factors, including the education program employed and the ethical behavior of the programmers. While recognized as clearly important, education will generally be insufficient in and of itself to prevent IP contamination, both malicious and non-malicious.

Another preventive solution is setting the requirement that only specific code may be used. Again, rules must be defined, documented and communicated on the acceptable practices and sources of code. A common criticism of this solution is that it may not generally apply. For example, programmers may find that it limits their choices and needs, and that acceptable alternatives become hindered by approval processes. Pressures of deadlines and deliverables create an atmosphere of tension between following the process and adhering to the rules. It is noteworthy that we have recently seen the emergence of successful commercial ventures that offer a database of IP-indemnified, pedigreed OSS.

Automated preventive solutions rely upon the detection and identification of external content immediately upon it being introduced into a project. Integration of the preventive solution within the development environment enables detection of external content, although it may not necessarily automatically identify the source of that content.

Detection can flag the introduction and optionally require the developer importing the code to annotate the source for future reference. Timely detection of the company's IP policy violations and possible immediate correction is included among the advantages of automated preventive solutions. Like corrective solutions however, preventive solutions do not address the related issues associated with determining software derivatives.

Summary

The software development industry is witnessing a transition, brought about by the explosive growth of OSS, code-search engines and outsourcing practices. The new order brought about by this transition carries certain IP challenges that must be effectively handled, otherwise the results could be catastrophic for a software company. IP policies must be set, monitored and enforced.

Mahshad Koohgoli is the CEO of Protecode Inc. (http://www.protecode.com), a software IP management company. Mahshad has been in the industry for a long time, has a BSc and a PhD from the University of Sussex in England. He holds various patents. He was the founder and CEO of Nimcat Networks, and founder of Spacebridge Networks and Lantern Communications Canada. He held senior roles in Newbridge, Bell Northern Research and Nortel.

Richard Mayer deals with software intellectual property issues as Vice President of Marketing for Protecode. He offers a breadth of experience and understanding of customer and technology challenges in the telecommunications and IT sectors. Prior to joining Protecode, Richard held senior marketing, product management and sales roles including an international posting in Nortel and JDSU. Richard has a degree in Computer Systems Engineering from Carleton University.

"A balanced approach to intellectual property rights is vital to economic growth."
Committee for Economic Development

In the Canadian copyright reform arena, the events of early December, 2007, changed everything.

In late November, it was widely anticipated that new copyright legislation would be introduced in the model of the controversial American Digital Millennium Copyright Act (DMCA). The bill was rumored to include harsh "anti-circumvention laws", which grant software distributors the right to seek legal remedies for circumvention of technological locks on content. In response, veteran Canadian copyright advocates issued an appeal to Canadians to take an interest in the bill and to call for fair and balanced copyright.

Canadian citizens answered that call in unexpected numbers, both online and offline. A Facebook group, called Fair Copyright for Canadians, grew to over 25,000 members within two weeks, and provided grassroots advocacy tools to citizens. A new website, called CopyrightforCanadians.ca, established itself as a centre for news on the bill and consumer advocacy. Using these tools, Canadians wrote letters, met with politicians, and demanded balance. With their words and their actions, not only did Canadians delay the introduction of the bill until next year, but they put copyright in the spotlight and showed legislators that fair and balanced copyright can capture the public imagination.

Industry Minister Jim Prentice now has the opportunity to hear from groups representing Canadian perspectives on copyright, hopefully resulting in a better and more balanced bill.

Open source developers have a unique relationship with copyright, with licences and practices enabling them to flourish in the copyright ecosystem. Open source software businesses are uniquely situated to comment on the copyright balance, since open source developers rely on copyright both for protection of their software, and for freedom to access the software of others. In the consultation process to come, the Canadian Software Innovation Association (CSIA) is in a position to be the voice of the Canadian open source industry.

The CSIA is a growing coalition of businesses, nonprofit organizations, individuals and user groups working in the business of free and open source software. The CSIA is an organization that advocates copyright laws in Canada that continue to support creativity and encourage innovation in the Canadian software industry while offering users necessary freedoms. The CSIA is being formed around a white paper, available at the CSIA's website (http://softwareinnovation.ca/whitepaper/). The white paper builds a case for Canada's need for copyright laws that support the open source model. The white paper offers three points to support this case: i) an account of Canada's open source business community and the economic contribution of open source to the marketplace; ii) an account of open source and its relationship to copyright law; and iii) an account of how anti-circumvention laws undermine open source software.

The white paper also offers a series of recommendations for addressing anti-circumvention laws. There is no such thing as a good anti-circumvention law, but if Canada is committed to implementing such laws, then it should do so in a way that minimizes potential disruption to the open source business model.

The CSIA offers guidance to the government on how to do so.

This short article offers a summary of the CSIA white paper, and issues a call to arms for open source developers: good laws don't just happen; law-makers need to understand the interest is in the balance. The CSIA offers a way to make your voice heard in copyright policy debates.

Economic Contribution of Open Source

Open source software (OSS) offers many practical and economic benefits to both open source businesses and users. Open source offers the public sector and small, medium, and large businesses alike, productivity and efficiency gains and other practical benefits such as specialization and scaling to meet a user's specific needs. Viewed more broadly, the open source model is a hotbed of innovation that contributes to Canada's economic well-being.

A 2005 Statistics Canada report discloses that open source software occupies a significant fraction of software operated by both private and public sector organizations. Over half (52.7%) of all public sector organizations reported using open source software. Among private sector firms, over two thirds (37.3%) of large firms reported using open source software, with less being reported by medium (16.5%) and small (9%) businesses (although these figures likely under-report open source usage through outsourced web services).

More work needs to be done to educate smaller businesses about the benefits and capabilities of open source software. Smaller businesses would benefit the most from the cost savings and flexibility that open source software delivers.

Open source consumer applications, such as the Firefox browser, Ubuntu Linux-based operating system, Apache web-server, MySQL database program, and OpenOffice productivity suite, are also increasing market share.

There are many economic advantages to using open source over proprietary software:

- Reduced Cost: considering the total cost of ownership of software, including the maintenance and support, in most cases OSS is less expensive than equivalent proprietary software

- Security: open source code is transparent to users, making software vendors more accountable, giving users and developers access to the source code and allowing them to identify vulnerabilities and provide or push for fixes in a timely manner

- Efficiency, scalability, and innovation: software upgrades can be controlled and determined by users and not the software vendor and the software may be easily customized to meet the specific needs of its users

Finally, open source encourages choices between vendors and programs and promotes interoperability for all users.

Although OSS is growing increasingly mainstream, Canada still lags behind other nations in take-up of OSS: studies show that firms use a larger percentage of OSS in the United States, Europe and the Pacific Rim. This suggests that closing the open source take-up gap offers Canadian firms the potential for significant productivity gains. Accordingly, the Canadian government should encourage the adoption of OSS to bolster Canadian productivity and economic growth.

This includes creating a business climate that is free of hurdles for open source implementation. Canada has a small but strong history of open source contribution. Nonetheless, Canada lags behind both the United States and the European Union in terms of open source development.

Open Source Software and Copyright

Copyright law is central to the philosophy and practices of open source developers. The public policy underlying Canada's copyright law seeks to balance the public's interest in rewarding authors for their creative efforts with the public's interest in access to those works. Open source developers similarly have interests on both sides of that balance.

Copyright law grants OSS developers exclusive rights in the computer programs they create, including the exclusive right to reproduce, distribute, and adapt and modify the software. Open source developers rely upon these exclusive rights to dictate the terms under which their software may be distributed and used. Open source developers collect these terms into standard licence agreements, simple legal documents that predictably and economically communicate the conditions under which the software may be used, adapted and distributed. The businesses and users that do not abide by the licence terms infringe copyright. The OSS copyright owner may seek remedies for infringement; the same way that the owner of copyright in a proprietary program might.

In return for the grant of copyright protection, the creator of a work is required to let the public engage in certain activities with the work created. This is often described as the "copyright bargain" in the sense that allowing access is the price the author pays for the benefit of copyright protection.

The development of open source software requires access to computer programs for many reasons, including the need to develop innovative extensions or extend the functionality of existing software, to undertake security research and to make code interoperable. Access to code is also required to research functionality, which includes reverse engineering code to identify non-patented inventions. Copyright law recognizes the value of these activities, and the need to place them beyond the absolute control of the copyright owner.

Technological Measures

Technical measures are ubiquitous, consumer-level digital technologies that allow copyright owners control over the distribution and use of software and digital content. Technical measures add a second layer of content protection that is independent of and in addition to copyright protection. The WIPO Internet Treaties oblige states to give legal protection to Technical measures. These legal protections, collectively known as "anti-circumvention laws", offer content distributors a third layer of protection.

Technical measures, like all technology, are fallible. Accordingly, anti-circumvention laws ineffectively prevent infringement but effectively deter legitimate uses of copyrighted works by law-abiding citizens, including open source developers.

Ten years ago this month, Canada signed the WIPO Internet Treaties. Now, the Canadian government has indicated that new copyright legislation will implement those treaties, implying that policy-makers will introduce anti-circumvention laws. Our recommendations for how policy-makers should implement the WIPO Internet Treaties without chilling innovation in the Canadian software development industry are as follows:

The best anti-circumvention legislation is no anti-circumvention legislation. Canada should not enact anti-circumvention laws. Canada is also under no international or domestic obligation to introduce such provisions. Anti-circumvention laws chill innovation and offer no further benefit that copyright law does not already offer.

If Canada does enact anti-circumvention laws, it should do so in a manner that avoids the mistakes that other nations have made in respect of such laws, and minimizes the potential for such laws to chill innovation. We suggest that:

- Canada must restrict the application of anti-circumvention laws to instances where a circumventor intends to infringe copyright; common sense dictates that legal activity must not be rendered illegal merely because of the presence of technical measures.

- Technical measures are seldom directed primarily towards content protection and, more often, they have anti-competitive objects. Anti-circumvention laws s should recognize this reality by balancing such protection with legal measures designed to protect the consumer.

- Proprietary software distributors often seek to lock in customers by minimizing interoperability while still satisfying the minimum needs of users. Open source developers, in contrast, seek interoperability among open source projects and their proprietary counterparts. Copyright's exceptions and limitations protect these efforts. Anti-circumvention laws should not frustrate them.

- Reverse engineering is a crucial tool for achieving interoperability and other legitimate ends. Canada must legislate the right of users to circumvent technological measures for non-infringing purposes.

- Secrecy does not protect systems effectively. Electronic security is improved through testing by the security community including academic security researchers and hobbyist programmers. Anti-circumvention laws chill security research since successfully breaching any form of technological measure protecting the underlying software raises the spectre of liability. The United States currently suffers from precisely this research chill. Canada has no such research chill and should not introduce one: bona fide security researchers must not operate under a cloud of liability.

- Even the most restrictive anti-circumvention laws permit circumvention for certain purposes. Tools to engage in such circumvention must be available in the marketplace to those who need them for legal use.

- Effective laws do not mandate the use of particular technologies. Technology mandates run the risk of losing relevance as technologies evolve and impose burdens on creator, user and distribution communities.

- Canada's general defense to infringement, fair dealing, suffers from deficiencies that undermines its utility, and so undermines the effectiveness of copyright law as a whole. Canada should expand fair dealing to address its current structural and technical inadequacies, and to clearly encompass reverse engineering and security research and access for purposes intended to allow interoperability.

Conclusion: A Call to Action

Copyright policy is innovation policy. Anti-circumvention laws would be harmful to innovation in Canada. Parliament should not pass anti-circumvention legislation.

If it chooses to do so, it must do so in a manner that does not undermine innovation policy in Canada.

There are many ways that concerned citizens and businesses can be active in the copyright reform process, especially now that a window is open for Canadian voices to be heard. The CSIA is a group representing stakeholders in the Canadian software industry. The CSIA is working to ensure that copyright legislation in Canada continues to support creativity and encourage innovation in the Canadian software industry. Since CSIA members use new modes of peer production, their position in this area differs from other software industry groups, and will provide a crucial voice in the consultation process. There is power in numbers. By joining the CSIA, you help to ensure that the voice of open source is heard.

Other good ways to get involved include joining 25,000 other Canadians in the Fair Copyright Facebook group (http://facebook.com/group.php?gid=6315846683) and adding your name to one of two petitions at Online Rights Canada (http://www.onlinerights.ca/get_active/copyright_pledge_petition/) and Digital Copyright Canada (http://digital-copyright.ca/petition/). It is also very effective to write to your MP, the Minister of Heritage, the Minister of Industry, and the Prime Minister. CopyrightforCanadians.ca offers a number of tools for helping you do just that.

Since much of the motivation for copyright reform in Canada comes from governments, organizations and businesses located outside our borders, it is particularly important in this context that Parliament carefully evaluate proposed changes in light of their impact on Canada's interests.

Where the interests of Canada's creative and innovative industries differ from those of foreign governments and lobbies, Parliament must act in the interests of Canadians, even where this may be viewed with disapproval by outside forces.

Finally, copyright law is designed to foster innovation and productivity. In order to do this, as in other areas affected by copyright, the law must maintain the balance between the interests of copyright owners and users of copyrighted works. In the software innovation context, this means granting copyright owners appropriate protections while ensuring access for innovative open source developers.

The events of December, 2007 provided evidence of Canadians' interest in the issue of fair and balanced copyright laws. As individuals, they have affected the legislative process and injected the public interest into the conversation. Now is the time for software innovators to state their position as well, while the window is open for their view to be heard.

Andy Kaplan-Myrth is the Manager of the Law & Technology group at the Faculty of Law at the University of Ottawa and an Associate of the Canadian Internet Policy and Public Interest Clinic. As a joint Project Lead with Creative Commons Canada, Andy speaks on open source, free culture and the sharing economy and promotes the use of OSS whenever the opportunity presents itself.

Contacts

David Fewer, Staff Counsel, CIPPIC
dfewer@uottawa.ca

CSIA questions, comments, or to join:
info@softwareinnovation.ca

"Language is a process of free creation; its laws and principles are fixed, but the manner in which the principles of generation are used is free and infinitely varied."
Noam Chomsky

The objective of this article is to:(i) extend the discussion of licensing to non-software assets and (ii) provide an introduction to rights expression languages (RELs). Licensing is not limited to software. We can associate a license with any kind of asset that holds intellectual value, and can thus be turned into a source of revenue. Here, our interest is on information assets, which include software and software components, but also services, processes, and content. For instance, a song that a user downloads from iTunes is an information asset. So is a web service such as the Google Maps API (application programming interface).

Licensing and DRM

We begin the discussion by defining two key terms, licensing and digital rights management (DRM). Licensing is a fundamental way of controlling the distribution of information assets, and underlies the design of business relationships and strategies. Licensing principles reflect the overall business value of assets to their producers and consumers. Also, licensing is often used to protect the intellectual property rights (IPR) of the producers of an asset. Licensing is both a source of revenue and a strategic tool.

In their book Digital Rights Management: Business and Technology, Rosenblatt, Trippe, and Mooney define DRM as an umbrella term referring to the collection of technologies (hardware, software, and services) that govern the access to information assets through associated rights, and controls their distribution (http://tinyurl.com/39kav4).

The foundation of DRM technology relies on our ability to represent the rights over digital assets. RELs represent the rights over assets in a machine-understandable way (http://www.loc.gov/standards/relreport.pdf). RELs describe different aspects of usage control, payment, and access, for a digital access environment.

According to Parrott, a REL consists of four components (http://xml.coverpages.org/RLTC-Reuters-Reqs.pdf):

- Subjects, the actors who perform some operation or action
- Objects, the content against which a subject wants to perform an operation
- Operations or what the subjects wants to do to the object
- A set of constraints or conditions under which an operation can be performed

These components and their relations support a range of models, each describing a way of applying digital rights. In general, a REL expresses the rights of an information asset either in some form of logic or in an XML-based language.

To illustrate these concepts, let us assume that a user wants to download a song from the iTunes store and play it on his iPod. The subject is the user, the object the song, the operation to play the song, and the constraints are that the user has to pay 99 cents for the download and cannot share the song with his friends.

Rights Expression Languages

What follows is a brief history of RELs.

A pioneering formal language called DigitalRights describes a mathematical model of simple licenses that consists of payment and rendering events and a formal representation of licenses (http://www.ldl.jaist.ac.jp/drcp/doc/dr-models.pdf). LicenseScript is a logic-based REL (http://www.ub.utwente.nl/webdocs/ctit/1/000000a1.pdf).

Logic-based RELs express general prepositions of a permissive or obligatory (restrictive) statement. However, these languages cannot express a finer level of granularity of the assets, actors, or actions involved. Logic-based RELs cannot interoperate with other types of RELs.

XML-based RELs support interoperable ways of expressing the rights of an information asset. An XML-based REL allows asset producers to specify flexible expressions. The Extended Rights Markup Language (XrML, http://www.xrml.org) and the Open Digital Rights Language (ODRL, http://odrl.net) are two XML-based RELs which have gained international recognition and are widely used in industry.

XrML is the basis for the REL of the MPEG-21 multimedia framework. It focuses on the license through which a rights holder confers usage rights to a consumer. A license can be digitally signed by the rights holder, now also referred to as the issuer, to confirm that the holder grants the rights contained in the license. An XrML license contains one or multiple grants and the license issuer. A grant is the element within the license that authorizes a subject to exercise a right on some object under some constraints. Note that the actual terminology used by XrML is slightly different from this.

ODRL is an open standard language for the expression of terms and conditions over assets in open and trusted environments. ODRL consists of an expression language and a data dictionary. The expression language defines basic terms of rights expressions and their organization using a set of abstract concepts. The data dictionary defines the semantics of the concrete terms used to express an instance of a rights specification.

ODRL is based upon an extensible model for rights expression, and defines the following three core entities and their relationships:

- Assets, the objects being licensed
- Rights, the rules concerning permitted activities, the constraints or limits to these permissions, the requirements or obligations needed to exercise the permission, and the conditions or specifications of exceptions that, if true, terminate the permissions and may require re-negotiation of the rights
- Parties, the information regarding the service provider, consumer, or broker

With these entities, ODRL can express offers (proposals from rights holders for specific rights over their assets) and agreements (contracts or deals between the parties with specific offers). ODRL supports the declaration of a wide range of expressions. It can also be extended to different types of domains. For example, we can use ODRL to specify that a consumer of a geocoding web service can only use this service in a non-commercial context, as well as the number of times the service can be accessed each day. ODRL has been published by the World Wide Web Consortium (W3C, http://www.w3.org/), and has received wide acceptance.

ODRL is supported by several industry consortia such as the Dublin Core Metadata Initiative (DCMI, http://dublincore.org/) and the Open Mobile Alliance (OMA, http://www.openmobilealliance.org/).

Two applications of ODRL are an ODRL profile of the semantics of Creative Commons (CC) licenses and the ODRL profile for services (ODRL-S). The core semantics of CC licenses have been expressed in ODRL. This profile supports extensions to these semantics, and defines an XML Schema (http://odrl.net/Profiles/CC/SPEC.html). ODRL-S (http://tinyurl.com/ypn5pu) is an extended version of ODRL to express clauses for service licensing, creating a machine-understandable service license.

Conclusion

Information assets are usually accompanied by a license that describes the terms and conditions on the use of this asset imposed by its producer. A license reflects the overall business value of the asset to its producers and consumers. The kind of rights vary based on the nature and context of the assets involved. For example, one of the rights for a multimedia asset is that consumers can play it. The concept of playing can not be directly applied to a web service asset.

Similarly, the rights governing the use of the interface and implementation of a web service are distinct. However, for multimedia or software assets we cannot make such a distinction. In this article we introduced the concept of licensing and RELs, and briefly described XrML and ODRL as the two most prominent RELs.

We thank Dr. Renato Iannella, NICTA, Australia, and Prof. Vincenzo D'Andrea, University of Trento, Italy, for their suggestions and comments.

G.R. Gangadharan is a doctorate student in University of Trento, Trento, Italy. His research interests include Free/Open Source Software Systems, Service Oriented Computing, Internet Software Engineering and Web 2.0, and Business Models of Software and Services.

Michael Weiss holds a faculty appointment in the Department of Systems and Computer Engineering at Carleton University, Ottawa, Canada, and is a member of the Technology Innovation Management program. His research interests include open source ecosystems, service-oriented architectures and Web 2.0, business process modeling, social network analysis, and product architecture and design. Michael has published on the evolution of open source communities and licensing of open services.

"The smartest one of us is all of us."

Anon

On June 29, 2007, the Free Software Foundation (FSF) issued the GNU General Public License Version 3 (GPLv3, http://www.gnu.org/licenses/gpl-3.0.html). The issuance of GPLv3 marked the end of a remarkable public consultation process aimed at revamping the license to address issues that the FSF (http://www.fsf.org) considers to be a threat to the Free Software movement, and to clarify issues that have been misunderstood or debated since the publications of GPL Version 2 (GPLv2).

This article introduces the rationale for changing the GPL and introduces the changes that affect patents, Digital Rights Management (DRM), license compatibility, and the linking issue.

Why Make Changes?

Despite its wide-spread use, the FSF concluded that GPLv2 had some shortcomings. In particular, the FSF was concerned that developments in the law since the publication of GPLv2, such as the increasing prevalence of software patents and laws prohibiting the circumvention of Digital Rights Management (DRM) technology, threatened the freedoms that it sought to protect and which were not adequately addressed in GPLv2.

In addition, it wished to increase the number of licenses with which the GPL is compatible so as to enlarge the pool of software code that can be combined with GPL-covered code. Finally, the FSF wanted to clarify certain provisions of the GPLv2 that it thought had either been misinterpreted or were incomplete.

Consultation and Drafting Process

On January 6, 2006, the FSF issued the first draft of GPLv3 and commenced a broad consultation process which included the formation of four discussion committees to consult and provide input with respect to each draft of GPLv3. The FSF also published a Rationale paper with each draft in which changes were summarized and annotated. It is clear from reviewing each of the drafts and Rationale papers that a tremendous amount of thought, debate and labour went into the drafting of GPLv3. The rest of this article summarized the key changes introduced in the GPLv3 and their effects.

Key Change #1: Patents

The FSF clearly understood that it could not use the GPL to eliminate software patents. In addition, it recognized that patent holders can be valuable contributors to GPL-covered computer programs, so long as they do not use their patents to take away the freedoms of others to use, modify, and distribute such programs. As a result, the FSF focused its attention on restricting users and distributors of GPL-licensed computer programs from using their patent rights against other users.

Explicit License

Unlike GPLv2, the FSF included an explicit grant of patent license in GPLv3. Based on earlier drafts, the FSF had intended to impose on all persons who distributed a copy of a GPL-covered program, including unmodified programs, the obligation to grant a license of any of their patents that would be infringed by the use of such program. However, many patent holders objected to the breadth of this policy and the FSF agreed to make a concession.

Instead, the grant of patent license applies only to those persons who distribute versions of the program that they have modified. It is important to note, however, that the patent license covers the entire work and not just the portions modified by the distributor. Patent holders who distribute modified versions of GPLv3-covered software will still have to institute and follow appropriate diligence procedures to ensure that they do not inadvertently grant a license to one or more of their patents.

Retaliation Clause

A patent retaliation clause provides that the license to use the computer program is terminated if a licensee brings a claim for patent infringement against the licensor. Retaliation clauses can be either narrow in scope, by restricting its effect to claims of infringement that relate to the licensed program, or broad and result in the termination of the license if the licensee brings a claim against the licensor for any patent infringement, regardless of whether or not it relates to the licensed program. GPLv3 adopts a narrow retaliation approach which is implemented through the interaction of Section 8 and Section 10 of GPLv3.

Extension of Patent License

Another instance in which software patents can threaten the freedoms which the GPL aims to protect occurs when a distributor of a GPL-covered computer program enters into an agreement with the holder of a patent that is essential to the use of such program whereby the patent holder covenants not to sue, or otherwise grants permission to, the distributor for its use and distribution of the program.

If such agreement does not extend to down-stream users of the program, the distributor will be in the position of being able to exercise the freedoms provided under GPL, but down-stream users will not.

To address this inequity, the FSF concluded that the recipient of a license to a patent that would otherwise be infringed by the use and distribution of a GPLv3-covered work must, as a condition to distributing such work, take certain actions to protect down-stream users. A beneficiary who "knowingly relies on" the patent license must either arrange to deprive itself from the benefit of the patent license for the affected program or arrange to extend the license to downstream recipients.

It should be noted that the words “knowingly relying on” were included in Section 11 to provide some comfort to those persons who have entered into blanket cross-licensing arrangements with others whereby they might not even know what patents have been licensed to them. As a third alternative, the beneficiary of the license may cause the source code to the program to be made available for anyone to copy, free of charge and under the terms of GPLv3.

Discriminatory Patent Licenses

In November 2006, Microsoft and Novell announced that they had entered into a broad business, technical and patent cooperation agreement under which Microsoft agreed not to assert its patents against users of Novell’s SUSE Linux Enterprise Server product; provided such users do not make or distribute additional copies.

By structuring the deal as a covenant not to sue, rather than as grant of a patent license to Novell, the parties sought to bypass the GPL provisions that would restrict Novell from redistributing GPL-covered programs in SUSE Linux Enterprise Server if it could not extend the benefit of a patent license to down-stream recipients.

In the opinion of the FSF, the Microsoft/Novell deal, and any similar arrangements, threaten the goals of the FSF and have the effect of making the affected GPL-covered program proprietary.

To address this threat, the FSF included in Section 6 of GPLv3 provisions to automatically extend patent licenses to all recipients of a GPLv3-covered program if the distributor grants a patent license to one of the recipients. Section 11 of GPLv3 also includes provisions which are intended to prevent other parties from entering into arrangements that are similar to Microsoft's deal with Novell. While there has been much debate regarding the effectiveness of such provisions, the uncertainty resulted in Microsoft stating that its deal with Novell would not extend to programs distributed under GPLv3.

Key Change #2: DRM

DRM is a term that is used to generally describe any system or method of protecting copyrighted or other proprietary material or data. While the FSF cannot outlaw the implementation of DRM-technology, in drafting GPLv3 it sought to prevent the use of GPL-covered computer programs from being used in such a manner.

The FSF often cites TiVo as an example of GPL-covered computer programs being used as part of DRM systems. TiVo, a personal digital recorder for television programs, includes GPL-covered computer programs. Pursuant to the terms of the GPL, TiVo provides the source code and users are permitted to make modifications to the programs. However, if a user attempts to run modified software on the TiVo hardware, the TiVo system will not operate.

To address its concern with DRM technology, Section 6 of GPLv3 provides that if any person distributes object code versions of a work licensed under GPLv3 as part of or for use with a “User Product” (as defined in GPLv3), such person must (with some exceptions) also make available “Installation Information”, which includes authorization keys or other information necessary to install and execute modified versions of the work in that User Product.

Section 3 of GPLv3 also contains provisions which seek to limit the application of various DRM and anti-circumvention statutes to GPLv3-covered software.

Key Change #3: Compatibility

While the GPL is the most widely-used open source license, there are many other licenses, not all of which are compatible with the various versions of the GPL. For a license to be compatible, a user must be able to combine the programs, or portions thereof, and distribute the combined work under the GPL.

If one cannot distribute the combined work under the GPL and still be in compliance with the other license, the two licenses are not compatible and the two programs cannot be distributed as a combined work.

In drafting GPLv3, the FSF wished to increase the number of open source software (OSS) licenses with which the GPLv3 is compatible without sacrificing the freedoms that the FSF seeks to preserve for users. To achieve this goal, Section 7 of GPLv3 permits additional permissions and a limited number of additional restrictions, or non-permissive provisions, to supplement the terms of GPLv3.

License incompatability is true even with GPLv2 and GPLv3, as these two versions of the GPL are not compatible with one another. Fortunately, many programs that are licensed under GPLv2 state that the program may be used under the terms of GPLv2 "or any later version". In such cases, the GPLv2-covered program can be combined with GPLv3-covered code and the combined work can be distributed under GPLv3. However, programs that are licensed under "GPLv2 only" may not be distributed as part of a combined work under GPLv3 unless the copyright holders of such program otherwise agree. For example, the Linux kernel is licensed under GPLv2 only.

Key Change #4: The "Linking" Issue

A computer program does not operate in isolation, and most programs are designed to utilize or link to other code, such as libraries. In discussions regarding the reach of the GPL, it has often been debated whether a covered work includes code that is linked to GPL-covered code.

For GPLv2-covered programs, the difficultly is a result of the absence of any operative language regarding linking and by confusing references to "derivative works" and "collective works". In GPLv3, these terms have been removed and a specific reference to linked code has been included. However, the FSF has not eliminated all ambiguities and it is not certain that the revisions have resolved the issue. It is clear that those who have used creative methods to avoid "contaminating" proprietary code, for example, by separately distributing libraries and add-ons, will have to re-evaluate their practices.

Conclusion

In revamping the GPL, the FSF addressed difficult, and often controversial, issues. The success of GPLv3 will be measured in large part by the rate at which it is adopted by developers and users of F/LOSS. While many development projects have already migrated to, or adopted, GPLv3, others appear to be taking a "wait and see" approach. For companies that include F/LOSS in products that they distribute, it is too soon to tell whether those that have previously distributed GPLv2-covered software will similarly embrace GPLv3-covered software. However, as this article explains, no decision should be made with respect to the use of GPLv3-covered software without first carefully considering the impact that the new provisions of GPLv3, such as the patent and DRM provisions, could have on one's business.

Eric is a lawyer in the Ottawa office of Fraser Milner Casgrain LLP where he also serves as Co-Chair of the firm's National Technology Transactions Practice Group. He is a frequent speaker and author regarding legal matters pertaining to technology-based companies.

"It's very important to remember that it's your intellectual property -- it's not your computer. And in the pursuit of protection of intellectual property, it's important not to defeat or undermine the security measures that people need to adopt in these days."

Stuart Baker, US Department of Homeland Security

New copyright legislation is on its way from the Canadian government, and may have been tabled by the time you are reading this. While we won't know exactly what is in the bill until it is tabled in the House of Commons for first reading, the Government has made many statements indicating that it intends to ratify the highly controversial 1996 WIPO treaties (http://www.digital-copyright.ca/node/4337). I believe it is important for open source developers and users to be aware of how some of the proposed changes may affect open source in Canada.

A recent ComputerWorld Canada article by Rafael Ruffolo quoted Barry Sookman, a lawyer specializing in intellectual property litigation with legal firm McCarthy Tétrault, as suggesting that the legal protection of technical protection measures proposed in the 1996 WIPO treaties would have no effect on open source (http://tinyurl.com/3azgus).

However, before any type of exclusive rights can benefit a software author, the right of owners of hardware must be protected such that it is possible that these people will choose our software. This puts software choice above all other concerns.

The Two Locks of DRM

When I am explaining Digital Rights Management (DRM) to politicians, I feel like Ralph Nader back in 1965. He explained that with an automobile accident there are two collisions: the car hits something, and the passenger hits the car. While automobile safety up to that point concentrated only on the first collision, it was quickly understood that safety features should concentrate on the second collision. This gave us dashboards that weren't made out of metal, seatbelts, air bags, and other such second-collision safety features. We have the same problem with DRM where policy makers think there is only one "digital lock" being discussed, when in fact there are two and it is the lock of which they are less aware that is the source of most of the controversy.

While the phrase DRM is used to refer to many unrelated things, the controversial form involves the use of a technical measure (most often cryptography) applied to two things: a digital lock on content, such as music, where that content can only be accessed with authorized tools containing the right decryption keys, and digital locks applied to access tools to disallow their owners/operators from controlling the tool. A tool can be software or a hardware/software bundle.

Both of these locks are harmful to software developers. The first lock is anti-competitive in that it forces people who wish to access encoded digital content to use specific brands of technology. If someone wants to access music downloaded from Apple's iTunes music store, they will be running Apple software. If someone wants to access encoded music downloaded from the new Industry-run Napster, they will need to be running Microsoft software.

While Sun has claimed that they have an open source DRM system with Open DReaM, this is only a distraction. It is not the file format or license of the software that determines what brand is required, but the encryption/decryption keys. The underlying software can be entirely open source, but your compiled version will not work because only those brands with the right decryption keys can access the content.

The second lock is far more controversial. The intention is to lock down the operations of the device such that the owner can not control it. The most obvious feature of such a system will disallow the owner from making their own software choices, thus disallowing them from choosing software with features more favourable to the user. In fact, software that allows user modification, one of the requirements for being open source, will never be allowed.

The more effective this technical measure, the less software choices hardware owners will be able to make, with the most effective technical measure disallowing the owner of the hardware from making any software choices. We see this today with hardware such as the TiVo where the BIOS is configured to only allow binaries which have been digitally signed by the manufacturer to run, meaning that the manufacturer makes all the software choices.

Private Ownership of Technology

Allowing private citizens to own and control their own information technology is critical. From a purely technological point of view, creativity and copyright infringement are identical technological acts. We record, edit and distribute content.

Any technology that attempts to reduce copyright infringement will only be able to do so by reducing the ability of private citizens to record, edit or distribute content. While the incumbent entertainment industry may like to reduce the competitive threat from newer creators, it should be obvious that it is bad for the economy overall to require that all creativity be authorized by the established content industry.

The dictionary defines capitalism as an economic system where the means of production is privately or corporatively owned. A basic part of ownership is the right to control what we own for lawful purposes. This policy attempts to ensure that the primary means of production and distribution of the primary outputs of the new economy cannot be privately owned and controlled. This should make us wonder exactly what type of economic system is being proposed by these policies?

Policy Concept Origins

In the early 1990's, governments were trying to understand the phenomena that was emerging from new communications technology and networks. In Canada we had the Information Highway Advisory Council (IHAC), and in the United States there was the National Information Infrastructure (NII) task force.

One of the sub-groups in the NII process was the Working Group on Intellectual Property Rights (WIPO, http://www.uspto.gov/go/com/doc/ipnii/). This working group brought together many of the established software and entertainment industry companies who saw this new technology as a threat to their existing businesses.

The basic thinking was: If new communications technology could be abused to infringe copyright, then private citizens should not be allowed to own and control this technology.

The NII Copyright Protection Act of 1995 proposed to severely regulate what private owners of technology were able to do with their technology. When this bill didn't pass in the United States, this thinking was brought to WIPO which created two treaties in 1996: The WIPO Copyright Treaty (WCT) and the WIPO Performances and Phonograms Treaty (WPPT). In 1998, the Digital Millennium Copyright Act (DMCA) was passed, and this law came into force in 2000.

It is important to know where the DMCA came from, and the type of pressure that is being exerted by the same US-based special interests on Canada to pass similar laws here. We also need to realize that the DMCA-style legislation suggested in the 1996 WIPO treaties is only one step, and that things can get much worse.

The Danger of One Lock

The target of old economy thinking was the private control of technology, and not content.

One such proposal is the Broadcast Flag (http://w2.eff.org/IP/broadcastflag) which would allow broadcasters to communicate without encrypting their content in any way, but would set a flag in the signal indicating whether or not people were allowed to record that broadcast. Technology manufacturers would be mandated to honour that signal, meaning that any technology capable of receiving broadcast signals could not be controlled by its owner. If the owner removed the foreign locks from their property, they would also be breaking the law.

As our desktop computer, home theater, and portable media increasingly merge, it becomes more difficult to separate general purpose computers from technology capable of receiving broadcasts. This would effectively mean that a broadcast flag type regulation would regulate the entire of the electronics and software industry, effectively banning anything that has owner modifiable/controllable parts inside.

There are also various proposals to close what is called the Analog Hole. One such proposal is to put watermark detection into any device capable of recording.

Here is a scenario that shows the weakness of the proposal: You are a parent and your child is taking their first steps. Unfortunately the radio is on, you are playing a music CD, or your child steps in front of the television and your camcorder refuses to record. The reality is that it is impossible for technology to be programmed to tell the difference between a pirate and a parent.

Petitioning Parliament

I created a petition to Parliament, called the "Petition to protect Information Technology Property Rights", to try to alert parliamentarians to this issue. This is a paper petition that follows all the rules of parliament so that it can be tabled in the House of Commons and demand a government reply. This petition has already been signed by hundreds of Canadians, and any additional signatures are always appreciated.

The title puts the focus on the technical measures applied to our information technology. It also clarifies that this technical measure is applied to information technology which neither the copyright holder nor the device manufacturer own.

The full text of the petition, and instructions on how to sign it and get it to the right people, can be found at http://www.digital-copyright.ca/petition/ict/. Early signatures have already been tabled in Parliament, but we need to continue to receive new signatures. This allows us to point politicians to a larger number of people who have signed, but also allows us to bring bundles of signatures to a larger number of politicians to table in the House of Commons. The more politicians that become aware of this issue, the more likely they are to protect our rights.

Talking with Fellow Creators

While it is critically important to talk to Canadian politicians and bureaucrats in the key government departments, we also need to open conversations with other creators.

Every creator whose form of creativity can be recorded, edited or distributed with the help of modern communications technology needs to understand the harm from governments revoking their personal control of this technology. Many creators are noticing changes in the marketplace for their creativity, and it is human nature to be frightened of the unknown. The initial reaction is always to try to stop change, even if that change turns out to be beneficial.

We need to ensure that creators understand the potential benefits to new communications technology, as well as the methods which reduce risks without harming the rights of all creators. Everyone needs to be made aware that DRM is not something that is applied to content, and thus should be seen as a choice made by copyright holders, but is primarily applied to devices which the owners should be legally protected to control.

Talking with Fellow Technicians

John Gilmore once said that, "the Net interprets censorship as damage and routes around it." This has unfortunately been interpreted by some technical people as suggesting that bad laws which regulate technology can be routed around as well.

While it is true that highly technical people may always be able to circumvent any locks applied by third parties. We need to be thinking in terms of what is commercially available to the average citizen. We can't write a subroutine to route around bad law, and it is critical for us to become politically engaged to ensure that our Parliament passes good laws when they regulate technology.

Conclusion

I believe that politicians have the best intentions when introducing and debating legislation. While our community may be highly technical, politicians are necessarily generalists and don't always understand the implications or unintended consequences of every bill before them. This makes it our responsibility as technologically informed Canadians to share our knowledge.

Recommended Resource

The Canadian DMCA: What You Can Do

(http://www.michaelgeist.ca/content/view/2431/125/)

The Digital Copyright Canada forum (http://digital-copyright.ca) was created to be a citizens forum to share ideas. We are not a formal organization or lobby group, but a place where we can help each other make sense of the changes that are underway and coordinate responses.

The Canadian Software Innovation Alliance has been formed to help give a voice to open source businesses in this area of policy (http://www.softwareinnovation.ca). Given that the Minister of Industry has claimed that all the CEOs that have contacted him are in support of anti-circumvention legislation, parliamentarians need to hear from us.

My hope is that we can make use of information sharing forums to ensure that the direction Canada takes on copyright faces forward into a future that fully recognizes the benefits to creativity and innovation of citizens control over communications technology and participation in new media.

Russell McOrmond is a self-employed Internet and F/LOSS consultant, joining the Free Software movement back in 1992. He is the policy coordinator for CLUE: Canada's association for Open Source, private-sector co-coordinator for GOSLING (Getting Open Source Logic INto Governments), and the host for the Digital Copyright Canada forum. Full contact information and links to these groups are at http://flora.ca.

The goal of the Talent First Network Proof of Principle (TFN-POP) is to establish an ecosystem anchored around the commercialization of open source technology developed at academic institutions in Ontario.

The priority areas are the commercialization of open source in:

- Mapping and geospatial applications
- Simulation, modeling, games, and animation
- Conferencing
- Publishing and archiving
- Open educational resources
- Social innovation
- Business intelligence
- Ecosystem management
- Requirements management

Expected Results

The TFN-POP is expected to:

- Establish a healthy ecosystem anchored around the commercialization of open source assets
- Maximize the benefits of the investment in the Talent First Network by the Ministry of Research and Innovation
- Accelerate the growth of businesses in Ontario that use open source assets to compete

Eligibility to Receive Funds

Individuals eligible to receive funds are faculty, staff, and students of universities and colleges in Ontario.

Budget and Size of Grants

A total of $300,000 is available. Applicants' requests should not exceed $30,000.

The TFN-POP may provide up to 50 percent of total project costs.

Criteria

Proposals will be judged against the following five criteria:

- Strength and novelty of open source technology proposed
- Extent of market advantage due to open source
- Project deliverables, likelihood that the proposed activities will lead to deliverable completion on time, and effectiveness of the plan to manage the project
- Track record and potential of applicants
- Extent of support from private sector

Application

The electronic version of the application received by email at the following address: TFNCompetition@sce.carleton.ca will be accepted as the official application. The email must contain three documents: a letter of support, project's vitals, and a project proposal.

Letter of support: (maximum 2 pages) a letter, signed by the person responsible for the Technology Transfer Office or Applied Research Office of the academic institution that proposes to host the project and the faculty member or student who will lead the project, must be included. This letter should describe the nature of the support for the project from the academic institutions, companies and other external organizations.

Project's vitals: (maximum 1 page) The project's vitals must include:

- Person responsible for applied research or technology transfer at the college submitting the proposal: name, mailing address, telephone number, and email address
- Project leader: name, mailing address, telephone number, and email address
- Team members: names, mailing addresses, telephone numbers, and email addresses
- Budget: Total budget, with TFN's contribution and that of other organizations
- TFN investment: TFN contribution broken down by payments to students, payments to faculty, and payments to project awareness activities

Project proposal: (maximum 5 pages) Project proposal must include the following:

- Project description: (maximum 1/2 page) Description of project.
- Benefits: (maximum 1/2 page) Description of the benefits of the proposed project, and an overview of the context within which the project is positioned
- Advantage: (maximum 1/2 page) Market advantage provided by open source assets used in the project
- Information on applicants: (maximum 1.5 pages) Background information to help assess the track record and potential of the people who are key to the project and the college
- Project plan: (maximum 2 pages) Description of the deliverables (what will be delivered and when); key project activities; nature of the involvement from companies, and other external organizations; and plan to manage the project

Evaluation & Deadline

Proposals will undergo review by the Expert Panel established by the TFN-POP. The Chair of the Panel may contact the applicants if required. A final decision will be communicated to the applicants within 30 days after the email with the official application is received.

There is no deadline. Applications will be evaluated on a first-come basis until the $300,000 available is committed.

Contacts

Luc Lalande: Luc_Lalande@carleton.ca

Rowland Few: rfew@sce.carleton.ca

About the Talent First Network

The Talent First Network (TFN) is an Ontario-wide, industry driven initiative launched in July 2006 with the support of the Ministry of Research and Innovation and Carleton University. The objective is to transfer to Ontario companies and open source communities: (i) open source technology; (ii) knowledge about competing in open source environments; and (iii) talented university and college students with the skills in the commercialization of open source assets.

David from CIPPIC writes: The Canadian Software Innovation Alliance (CSIA) is a coalition of open source businesses concerned about the potential negative impact of new, US-style digital copyright laws. Formed under the leadership of Bob Young, co-founder of Red Hat with the support of CLUE (the Canadian Association for Open Source), and facilitated by the Canadian Internet Policy and Public Interest Clinic (CIPPIC), the CSIA's mandate is to provide a voice for open source business in copyright policy debates.

We expect the imminent introduction of a new copyright reform bill. We understand this bill to be modeled upon the American Digital Millennium Copyright Act, and that the bill will have serious implications for open source developers. Open source software relies upon copyright law—both its protections, and exceptions. We want to ensure that any changes to Canada's copyright laws reflect the needs of the Canadian open source community. These important issues are highlighted in a CSIA white paper which explains the Canadian open source community's needs in copyright reform.

The Government of Canada is introducing these extreme laws, in part, because it claims that it has not heard from Canadian businesses that favour balanced copyright laws. We hope that you will answer this claim by joining this coalition of open source businesses and speaking out on the need for balanced copyright laws that support innovation in software. And please, spread the word—pass this invitation along to other businesses that you believe should support copyright policies that support open source software.

To join this coalition, simply contact info@sofwareinnovation.ca. To learn more about the CSIA, visit http://www.softwareinnovation.ca.

Editor: The article A Rallying Moment for Canadian Open Source Software in this issue of the OSBR is based upon this white paper. The entire white paper is available for download from the Software Innovation website.

Glenn from Toronto writes: With regards to the NOSI Primer mentioned in the November issue of the OSBR, it should be noted that an updated version of the Primer was released in October of 2007. The new version is available from http://www.nosi.net/projects/primer and contains case studies of open source usage. If you're curious on the scope of the new primer, I also recommend that you read the brief synopsis by the author Michelle Murrain as posted at the ICT Hub website created by LASA at http://www.icthubknowledgebase.org.uk/nosiprimer, whereupon she encapsulates the nature of the changes in the new updated Primer.

Thomas from Ottawa writes: As 2007 winds down, predictions about what 2008 will have in store are plentiful (where are the stock markets going, will there be a federal election in Canada, etc.). Along the same lines, the following article (http://www.networkworld.com/news/2007/112707-open-source.html) discusses trends for open source in 2008, which may be of interest to the readers of OSBR. Of course, we all know with hindsight how accurate (or not) these predictions are, but the article nevertheless raises some interesting points to ponder. In particular the prediction that Microsoft will become much more actively involved in open source will undoubtably be controversial, given the history of open source and the strong feelings about Microsoft and its business model held by certain parts of the technical community.

Library and Archives Canada launches the Government of Canada Web Archive

November 20

The Library and Archives of Canada Act received Royal Assent on April 22, 2004, allowing Library and Archives Canada (LAC) to collect and preserve a representative sample of Canadian websites. To meet its new mandate, LAC began to harvest the Web domain of the Federal Government of Canada starting in December 2005. As resources permit, this harvesting activity will be undertaken on a semi-annual basis. The harvested website data is stored in the Government of Canada Web Archive (GCWA). Client access to the content of the GCWA is provided through searching full text by keyword, by department name and by URL. LAC has implemented this first significant Canadian Web archive through the use of open source tools, developed by the International Internet Preservation Consortium (http://www.netpreserve.org), of which LAC is a member.

http://tinyurl.com/2fnvfv

An Open Challenge to the Open Source Community: Make it or Break it!

November 28, Toronto, ON

SQL Power is proud to present the Power*MatchMaker: Make it or Break it Contest, an opportunity for developers and database users across the globe to win cool prizes by helping us make the best Open Source Data Cleansing Tool available even better. Prizes include limited edition Power*MatchMaker T-shirts, USB Flash Drives, and of course bragging rights! Refer to the contest page for full contest rules and regulations.

http://www.sqlpower.ca/page/matchmaker_contest

January 16-18

McMaster World Congress

Hamilton, ON

The McMaster World Congress is an international business conference which has provided a collaborative framework bringing together academic researchers, business practitioners, management students and global thought-leaders to present, discuss and review the latest issues, trends, challenges and opportunities in the fields of corporate governance, intellectual capital and strategic business valuation.

http://worldcongress.mcmaster.ca/2008/Default.asp

January 22

Workshop on Open Source Best Practices

Montreal, QC

The commercial use of open source is hindered by many factors. These include a lack of integration with traditional requirements-driven product development approaches, licensing issues, a clash with existing corporate culture, and the perception that in order to benefit from open source you need to open your source to the outside world. The goal of this workshop is to bring together researchers and practioners with experience in open source adoption and value creation from open source, and to document the best practices.

http://www.carleton.ca/tim/events/wosbp2008/

January 23-25

Montreal Conference on eTechnologies

Montreal, QC

MCETECH2008 will feature a special track on open-source software for e-businesss, which brings an additional twist to the usual technical, organizational, and regulatory aspects of e-business. We also welcome contributions that deal with the extent to which open-source e-business software helps bridge the digital divide that exists between developed and developing countries.

http://www.mcetech.org/

Free/Libre Open Source Software: a Guide for SMEs

Editor: Carlo Daffara for the FLOSSMETRICS EU project

From the Introduction:

"Open source software is the most significant all-encompassing and long-term trend that the software industry has seen since the early 1980s"...Despite this situation, there is still a significant barrier in the adoption process for small and medium companies, both in terms of using F/LOSS internally and in creating products and services centered on F/LOSS products. The purpose of this report is to provide a simple and in-depth view of the fundamental aspects of F/LOSS, how to adopt it within a small/medium company, and how to build a sustainable business based on it.

http://guide.conecta.it/FLOSSguide.pdf

Free/Libre Open Source Software: Software Catalog

Editor: Carlo Daffara for the FLOSSMETRICS EU project

From the Introduction:

This software catalog is a companion document of the F/LOSS guide for small and medium enterprises prepared in the context of the FLOSSMETRICS EU project; it is based on the work of the OpenTTT project (http://www.openttt.eu), that helped in the identification of needs through a large number of interviews and audits within European SMEs interested in FLOSS.

http://guide.conecta.it/FLOSScatalog.pdf

When and How ICT Interoperability Drives Innovation

Authors: Urs Gasser and John Palfrey

From the Introduction:

The scope of our study is to understand the dynamics of interoperability in the information and communications technology (ICT) space, with a particular view toward its relationship to innovation. We have looked at many issue areas in the course of this research, though our focus has primarily been on three areas in detail: DRM, Digital ID systems, and Web Services. Within Web Services, we have dedicated the written case study to the emerging area of mashups. In addition to these focus areas, we also looked closely at (though have not written up as cases here) other issue areas within the ICT arena where interoperability has been a major topic. These secondary areas of interest include the widely publicized matter of document formats for word processing applications and the like; other aspects of the digital media space outside of DRM struggles, such as digital video formats and digital data carriers; eCommunications such as instant messaging and content on mobile devices; and other aspects of the Web services environment, such as content syndication. Based upon these cases, we have extracted general principles from the case studies where we think stable, reliable patterns emerge.

http://cyber.law.harvard.edu/interop/pdfs/interop-breaking-barriers.pdf

The goal of the Open Source Business Resource is to provide quality and insightful content regarding the issues relevant to the development and commercialization of open source assets. We believe the best way to achieve this goal is through the contributions and feedback from experts within the business and open source communities.

OSBR readers are looking for practical ideas they can apply within their own organizations. They also appreciate a thorough exploration of the issues and emerging trends surrounding the business of open source. If you are considering contributing an article, start by asking yourself:

1. Does my research or experience provide any new insights or perspectives?

2. Do I often find myself having to explain this topic when I meet people as they are unaware of its relevance?

3. Do I believe that I could have saved myself time, money, and frustration if someone had explained to me the issues surrounding this topic?

4. Am I constantly correcting misconceptions regarding this topic?

5. Am I considered to be an expert in this field? For example, do I present my research or experience at conferences?

If your answer is "yes" to any of these questions, your topic is probably of interest to OSBR readers.

When writing your article, keep the following points in mind:

1. Thoroughly examine the topic; don't leave the reader wishing for more.

2. Know your central theme and stick to it.

3. Demonstrate your depth of understanding for the topic, and that you have considered its benefits, possible outcomes, and applicability.

4. Write in third-person formal style.

These guidelines should assist in the process of translating your expertise into a focused article which adds to the knowledgable resources available through the OSBR.

Upcoming Editorial Themes

January 2008	Interoperability
February 2008	Data
March 2008	Procurement
April 2008	Communications
May 2008	Enterprise Readiness
June 2008	Security

Formatting Guidelines:

All contributions are to be submitted in .txt or .rtf format and match the following length guidelines. Formatting should be limited to bolded and italicized text. Formatting is optional and may be edited to match the rest of the publication. Include your email address and daytime phone number should the editor need to contact you regarding your submission. Indicate if your submission has been previously published elsewhere.

Articles: Do not submit articles shorter than 1500 words or longer than 3000 words. If this is your first article, include a 50-75 word biography introducing yourself. Articles should begin with a thought-provoking quotation that matches the spirit of the article. Research the source of your quotation in order to provide proper attribution.

Interviews: Interviews tend to be between 1-2 pages long or 500-1000 words. Include a 50-75 word biography for both the interviewer and each of the interviewee(s).

Newsbytes: Newsbytes should be short and pithy--providing enough information to gain the reader's interest as well as a reference to additional information such as a press release or website. 100-300 words is usually sufficient.

Events: Events should include the date, location, a short description, and the URL for further information. Due to the monthly publication schedule, events should be sent at least 6-8 weeks in advance.

Questions and Feedback: These can range anywhere between a one sentence question up to a 500 word letter to the editor style of feedback. Include a sentence or two introducing yourself.

The Talent First Network program is funded in part by the Government of Ontario.

http://www.carleton.ca/tim

The Technology Innovation Management (TIM) program is a master's program for experienced engineers. It is offered by Carleton University's Department of Systems and Computer Engineering. The TIM program offers both a thesis based degree (M.A.Sc.) and a project based degree (M.Eng.). The M.Eng is offered real-time worldwide. To apply, please go to: http://www.carleton.ca/tim/sub/apply.html.