OSBR.CA

The Open Source Business Resource

Editorial
Dru Lavigne, Michael Weiss

An Introduction to Open APIs

Mapping Mashup Ecosystems
Michael Weiss

Licensing of Open APIs
G.R. Gangadharan

Using JavaScript Toolkits to Create Rich Internet Applications
Owen Byrne

Measuring Modularity in Open Source Code Bases
Steven Muegge, Roberto Milev

Torys Technology Law Speaker Series: Open Source Licenses and the Boundaries of Knowledge Production
Byron Thom

Upcoming Events

Newsbytes

Contribute

APRIL
2009

APRIL 2009

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

Editorial
Dru Lavigne and Michael Weiss discuss the editorial theme of Open APIs. 3

An Introduction to Open APIs
Carleton University students provide a glossary of terms associated with Open APIs and present some resources that discuss their business value. 5

Mapping Mashup Ecosystems
Michael Weiss from Carleton University describes a research framework to examine the structure of the mashup ecosystem and its growth over time using network analysis to obtain key characteristics of the ecosystem and identify significant ecosystem members and their relationships. 8

Licensing of Open APIs
G.R. Gangadharan, a research scientist at the Novay in the Netherlands, provides an overview of open API licensing, provide examples from current open APIs, and briefly discusses open licensing of open APIs. 13

Using JavaScript Toolkits to Create Rich Internet Applications
Owen Byrne, Senior Manager of Travelpod Labs, compares the features of the most commonly used JavaScript Toolkits used to create rich Internet applications and discusses how freely available toolkits are able to compete against proprietary alternatives. 17

Measuring Modularity in Open Source Code Bases
Steven Muegge and Roberto Milev of Carleton University describe a method for examining the evolving modularity of large-scale software systems and introduce a new modularity metric for comparing code bases of different size. 21

Torys Technology Law Speaker Series
Byron Thom, a student at at the University of Ottawa's Faculty of Law, summarizes a recent Torys Technology Law Speaker Series presentation which introduced a novel interpretation of copyright in the age of OSS using a spatial framework to deal with open source licenses. 27

Upcoming Events
Canadian open source events at a glance. 30

Newsbytes
Whats new and notable in the world of open source. 32

Contribute 34

A few short years ago, the term "Internet" reflected the technical savvy sitting at a workstation reading email or using a search engine to find data. Today, people of all ages are using all manner of devices to: obtain public transit directions with Google Maps, share photos using Flickr and videos using YouTube, Tweet their whereabouts, meet new friends through Facebook, and perform countless other activities which have quickly become ubiquitous to every day life.

This new generation of online activities is the result of open APIs, mashups, and rich Internet applications. These concepts are the focus of the April issue of the OSBR. The authors have done an excellent job of taking the editorial theme of "Open APIs" from the mysterious realm of programming into their applicability to daily life and business.

As always, we encourage readers to share articles of interest with their colleagues, and to provide their comments either online or directly to the authors. We hope you enjoy this issue of the OSBR.

The editorial theme for the upcoming May issue of the OSBR is "Open Source in Government" and the guest editor will be James Bowen from the University of Ottawa. Contact the editor if you're interested in a submission for this issue.

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 andDNSStuff.com and is the author of the books BSD Hacks and The Best of FreeBSD Basics.

Websites and applications on the Web serve significant volumes of data. In the past, this data was hidden behind web pages that only humans could read. This made it difficult for others to reuse this data in other applications, usually involving tedious and volatile web scraping. It also required users to come to your website to access the data, which limited its reach.

The editorial theme of this issue, "Open APIs", presents a recent solution to this problem: providing an open API to your website. The articles in this issue show how companies can make their data more accessible and extend their reach through open APIs. They also discuss the issues and techniques related to the provision and use of open APIs.

This issue has four articles related to the open API theme and two regular contributions. The first article, An Introduction to Open APIs, was written collectively by the students of a course on Web 2.0: Collective Web, which is offered as one of the courses within the Technology Innovation Management (TIM) program at Carleton University. It provides an introduction to the terminology of open APIs and mashups, and discusses the business reasons for opening an API.

The second article, Mapping Mashup Ecosystems, by Michael Weiss of Carleton University, takes an ecosystem perspective on mashups and open APIs. It presents a method for mapping the mashup ecosystem, and discusses the managerial insights we can gain from this analysis.

The third article, Licensing of Open APIs, by G.R. Gangadharan at the Novay Telematica Institute in The Netherlands, examines a topic whose importance will only grow with time, as open APIs and mashups get more ingrained in the way we develop software. It deals with the important issue of protecting the intellectual property contained in open APIs. It gives an overview of open API licensing and provide examples from current open APIs. It also discusses strategic issues of licensing open APIs.

The fourth article, Using JavaScript Toolkits to Create Rich Internet Applications, is written by Owen Byrne, co-founder and original developer of digg.com, and currently Senior Manager of Travelpod Labs in Ottawa. It discusses the selection and use of sophisticated JavaScript toolkits like Prototype and jQuery, which are essential frameworks for writing Rich Internet Applications (RIA) and mashups. It informs us on the importance of open source alternatives to proprietary frameworks for constructing RIAs. It also describes the author's experience in using these toolkits in building a meta-search application for tripadvisor.com.

The fifth article, Measuring Modularity in Open Source Code Bases, by Steven Muegge, a faculty member in the TIM program, and Roberto Milev, a recent graduate from the TIM program, examines how modularity evolves in open source software systems. It provides initial evidence that as a large software system evolves, major architectural changes will first lead to a decrease in modularity, and are then followed by refactoring activities, which increase modularity.

The sixth article, Torys Technology Law Speaker Series: Open Source Licenses and the Boundaries of Knowledge Production, by Byron Thom, a student at the Law Faculty of the University of Ottawa, summarizes a recent lecture given by Michael Madison, Associate Dean for Research and Professor of Law at the University of Pittsburgh School of Law.

I hope you enjoy learning more about this editorial theme as much as we enjoyed putting this issue together. Please feel free to contact the authors or the editors for questions, insights, or comments on this important topic.

Michael Weiss

Guest Editor

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

"The API has been arguably the most important [...] thing we've done with Twitter. It has allowed us, first of all, to keep the service very simple and create a simple API so that developers can build on top of our infrastructure and come up with ideas that are way better than our ideas, and build things like Twitterrific, which is just a beautiful elegant way to use Twitter that we wouldn't have been able to get to, being a very small team."

Biz Stone, co-founder of Twitter
http://www.readwriteweb.com/archives/twitter_open_platform_advantage.php

This article provides a glossary of terms associated with open APIs which can serve as an introduction to the other articles in this issue of the OSBR. We then discuss the business opportunities that can be created through an open API and provide video and text resources which present further thoughts on the business value inherent in open APIs.

Glossary of Terms

Here we present definitions for the most commonly used terms associated with open APIs.

AJAX: asynchronous JavaScript and XML. Enables updates to a web page without requiring the browser to reload the page. The rationale behind AJAX is that often only a small portion of the web page changes. Web pages using AJAX are significantly more responsive than pages that do not.

API: application programming interface. Wikipedia defines an API as "a set of routines, data structures, object classes and/or protocols provided by libraries and/or operating system services in order to support the building of applications" (http://en.wikipedia.org/wiki/Api).

Collective intelligence: collective intelligence is the information and insights that can be extracted from the collective set of interactions and contributions made by a user community, and the use of this intelligence to act as a filter for what is valuable to the users. Collective intelligence emerges from user-contributed content and the process of sense-making.

Generativity: the capacity of a system to produce unanticipated changes through unfiltered contributions from broad and varied audiences. Technological generativity describes the quality of the Internet that allows people unrelated to vendors to produce content in the form of applications through mashups and user-contributed content in the form of wikis and blogs. The first generation of the Internet was a non-generative system whose content was controlled by a small number of parties. With Web 2.0 technologies and practices, the generative potential of Web has reemerged, allowing users to participate and collaborate in the creation of its content.

Mashup: mashups combine data and services provided by third parties through open APIs, such as Google Maps and Flickr, as well as internal data sources owned by users. Mashups are an example of recombinant innovation. Mashups can be implemented directly within the client browser or on a server. Client-side mashups often access open APIs through AJAX. Wikipedia describes and provides examples of different types of mashups (http://en.wikipedia.org/wiki/Mashup_(web_application_hybrid)).

Network effect: network effects, also known as network externalities, occur when the value of a good, service or a shared resource is affected by the number of its users. Network effects can be positive when the value of the good increases with the number of users.

An example of a positive network effect is seen when more people use a type of credit card, causing more merchants to accept it. Network effect can also be negative when the value decreases as the number of users increase. A highway, jammed with cars reduces its value to each commuter as traffic slows, is an example of a negative network effect.

Open API: an open API gives users access to the open content data or services of an information technology (IT) platform. A well-known example is the Google Maps API which generates maps for a given location, and its output can be combined with other data and services into mashups. Open APIs provide users with an innovation toolkit in the sense of the user innovation paradigm. The P2P Foundation (http://p2pfoundation.net/Open_API) provides a further discussion on the importance of open APIs.

Open content: one [illegible]he defining characteristics of Web 2.0 is openness. Open content is published in a format that explicitly allows copying and modification. The Creative Commons (http://creativecommons.org/) provides a choice of licenses which are often used to describe the rights associated with open content.

Recombinant innovation: describes a view of innovation as a process through which new ideas emerge as the combination of existing ideas. This process can shorten the learning curve as it combines known elements in novel ways. Recombinant innovation allows innovators to share past experience and provides a diversity of problem solving frames.

Sensemaking: the capacity for making sense of complex sets of information during a situation in which new problems, opportunities, or tasks present themselves, or old ones resurface. Sensemaking aims to help people act in an informed and effective manner. Mashups and other Web 2.0 technologies support the sensemaking process by enabling collaboration, visualization, and casual interaction with data sources through which users can gain insights into the structure and interpretation of the data.

User-contributed content: Web 2.0 practices and technologies have empowered users to participate and collaborate in the creation of content. This content created by users is the basis for the existence of social networking websites and portals. Similarly, websites like Flickr and YouTube primarily depend upon user uploaded content and provide a framework to categorize the content using user-generated tags. Contributions such as photos, reviews, ratings, and lists of friends are considered to be active. Contributions in the form of behavioural data such as clickstreams, page views, and purchases as well as resources such as computing capacity are considered to be passive.

User innovation: traditionally, product development has been company-centric. In this model, the interface to the customer is the product prototype and feedback on how well customer needs are being met is obtained late in the product development cycle. In user innovation (http://en.wikipedia.org/wiki/User_innovation), the locus of innovation shifts from the company to the customer. The new interface to the customer is now a solution platform that customers can adapt to their needs using innovation toolkits. Open APIs can be considered innovation toolkits.

Web 2.0: a phenomemon that forms the basis of the next generation of the Internet and that manifests itself in terms of user-contributed content, openness as seen in open APIs and open content, and network effects.

Web scraping: an approach to extract structured information from websites that do not offer an open API to provide data access. An API to a website created through web scraping is also known as an implicit API. An example of web scraping is HousingMaps (http://housingmaps.com), the first Google Maps mashup which mixed data from Craigslist with Google's maps. At the time, neither Craigslist or Google Maps provided an open API to their services, so the creator of HousingMaps had to resort to web scraping to extract the data from these sites.

Widgets: also known as gadgets, these are small, reusable components that allow content from multiple sources to be easily integrated into websites without programming. For example, a widget can provide access to a user's Twitter feeds on her home page (http://widgetbox.com/widget/twidget). The term widget can be used to refer to a mashup that composes only one open API. Most early mashups were widgets, and widgets continue to represent a significant percentage of mashups. To integrate a widget into your website, you only need to paste an URL or a piece of JavaScript into the HTML code of your page.

Importance of Open APIs

As mentioned in the introductory quote by Twitter co-founder Biz Stone, opening an API to an application creates opportunities for external innovation. Giving third-party developers programmatic access to an application allows them to add value in unanticipated ways, and adds resources to your development effort that you would not otherwise have access to. You can thus tap into the long tail (http://en.wikipedia.org/wiki/The_Long_Tail) of underserved users who write their own applications to meet their specific needs, if given the opportunity.

With external development, the risk of development is carried by others, but you can nonetheless reap the benefits from successful innovations. This is what Google does when it gives users access to its vast computing infrastructure when providing such services as Google Maps, or any of its other (http://www.programmableweb.com/apitag/?q=google&sort=date).

When you open an API, follow established standards where they exist. Jakob's law (http://www.useit.com/alertbox/20000723.html) implies that the largest opportunity to increase site traffic is to give users an easy way of integrating your content into their sites. This can take the form of a widget or an open API which other developers can use to build widgets and mashups that leverage your API. Increased traffic provides opportunities to monetize your application through other avenues, such as advertising.

If your application or service collects information from users, such as photos or profiles, users expect to get access to their information through an open API. They don't want to be locked into a particular service, and will base their decision on which service to use, in part, on the existence of an open API.

The glossary definitions were contributed by the students of the SYSC 5801 course on Web 2.0: Collective Web (http://www.sce.carleton.ca/faculty/weiss/courses/SYSC5801/SYSC%205801%20Outline.pdf), a graduate course taught in the TIM program. This glossary will be part of a wiki book on Web 2.0 that the students have written collectively.

"A key advantage of the advent of open APIs is that many people can simultaneously tackle a particular problem by working on their own version of a mashup."

Palfrey & Gasser
http://cyber.law.harvard.edu/interop/pdfs/interop-mashups.pdf

Mashups enable users to "mix and match" data and user interface elements from different online information sources to create new applications (http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=4620093). The creation of mashups is supported by a complex ecosystem of interconnected data providers, mashup platforms, and users. In our recent research, we examined the structure of the mashup ecosystem and its growth over time. The main contribution of our research is a method for the analysis of mashup ecosystems. Its novelty lies in the development of techniques for mapping the mashup ecosystem, and the use of network analysis to obtain key characteristics of the ecosystem and identify significant ecosystem members and their relationships. In this paper, we summarize the key steps of our analysis method, describe the members of the mashup ecosystem, and discuss the managerial implications of our analysis.

Mapping the Mashup Ecosystem

There are many public sources that provide information about open APIs, mashups, and associated platforms. One source, ProgrammableWeb.com, lists APIs and mashups by date of introduction, and provides a profile of each. It also categorizes APIs and mashups through a provided taxonomy and through tags that users can associate with the entries. In addition, the site provides information on mashup tools.

Since the contents of the site are user-contributed, not all APIs and mashups in existence are indexed. However, the ProgrammableWeb is probably the most widely recognized mashup directory, and its contents can be considered representative of the state of the mashup ecosystem. Thus, while our analysis is likely to underestimate the total size of the mashup ecosystem, it can be expected to accurately represent the relations between ecosystem members.

First, we extracted time-stamped information on when APIs were introduced and when mashups were created. Next, we captured the relationships between mashups and APIs in an affiliation network. Originally developed for representing teams and their membership by Uzzi et al. (http://www.kellogg.northwestern.edu/faculty/uzzi/ftp/Uzzi_EuropeanManReview_2007.pdf), the links in the affiliation network for the mashup ecosystem indicate which APIs are used in which mashups. Affiliation networks lend themselves to a visual analysis of the network data and many relationships only become apparent by visualization. Visual observations then direct the further analysis as to what aspects of the network to study.

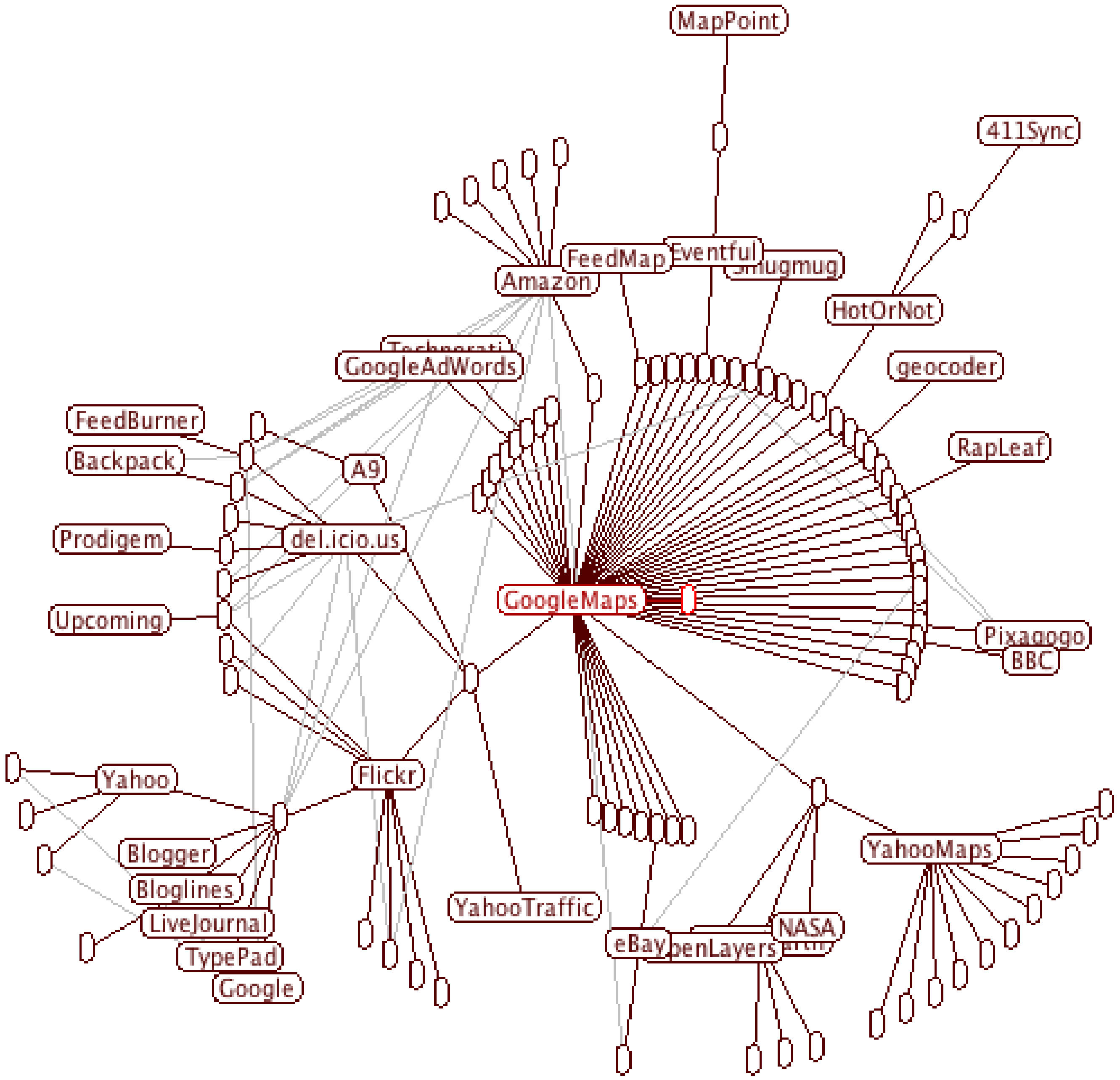
Figure 1 shows a snapshot of the mashup ecosystem using data based on the first month of records on the ProgrammableWeb site. Only the names of APIs are shown to keep the diagram readable, mashups are shown as unnamed nodes, and a link between a mashup and an API indicates that the mashup uses the API. Even at this early stage, some of the most well-known open APIs are already prominently positioned in the network. We find Google Maps at the center and other prominent APIs, such as Flickr, Amazon, Yahoo Maps and del.icio.us, along the first ring around the center.

Named nodes at the periphery of the graph represent APIs that have been used less frequently. Similarly, there are many one-feature mashups, also known as widgets or badges, that combine only one external API with internal data.

Members of the Mashup Ecosystem

Initially, there were two types of members within the mashup ecosystem: i) data providers that release open APIs, such as Flickr or Google; and ii) users and developers creating mashups. The only way for users to create mashups was by manually combining open APIs exposed by data providers. In some cases, data providers aggregate the data offered by other providers. For example, Google Maps obtains its raw map data from a number of geographic data services. Access to those data providers is often not directly accessible through open APIs, so an API such as Google Maps is not itself a mashup.

Figure 1: Snapshot of Mashup Ecosystem



Somewhat surprisingly, there are only a few mashups that offer their own APIs. We believe that the rationale for this is a combination of licensing issues and business reasons.

Open APIs and Mashups

Throughout the observation period, the data shows a consistent growth in the number of open APIs and mashups. Each day on average 0.70 new APIs were defined, from which users created an average of 3.10 mashups each day. On average, there are 4.41 mashups to each API. Such linear growth was also observed in other types of networks such as collaboration networks (http://arxiv.org/abs/cond-mat/0104162/). However, the distribution of mashups over APIs is far from uniform, following a power law with a characteristic "long tail" (http://en.wikipedia.org/wiki/The_Long_Tail). Some APIs enjoy significantly greater popularity than others.

One explanation for the long tail is competition between APIs that offer the same type of service. For example, multiple APIs provide map services such as Google Maps, Yahoo Maps and Microsoft's Virtual Earth. Choice requires user selection, and users will initially prefer some APIs over others. The more they select one API, the more likely it will be selected in the future. The result is that, eventually, one API will be significantly more popular than others. However, it also implies that a small number of APIs, the keystones of the ecosystem (http://williamkuo.bokee.com/inc/HBR-Strategy_as_Ecology.pdf), provide the basis for the majority of mashups, and all other APIs are only used in certain application niches.

Complementary Nature of APIs

Open APIs are the components of mashups, and as such provide value to users by themselves. However, their value increases when other data providers offer complementary APIs that extend their functionality or allow them to be used in new contexts. For example, Flickr complements Google Maps, because it allows images about a given location to be shown on a map. In fact, the combination of these services was so compelling that both Flickr and Google decided to offer new services to show images on a map. APIs that have many complementary APIs are more attractive to users, and will be selected more often for inclusion in a new mashup.

Starting from the affiliation network that shows how APIs are used by mashups, we created a network that shows just the APIs, with links indicating which APIs are used together in a mashup. The network has a core that consists of a small number of highly connected APIs that are used by many mashups, and more specialized APIs that are linked to the core. The nodes in the core of the network are APIs which attract keystone data providers as well as niche data providers as complementors. In part, this is certainly due to an accumulation of coordination knowledge. As APIs are used together, users build up an experience base on how to integrate them, making those APIs more popular.

Mashup Platforms

As the number of APIs, and thus the complexity of selecting mashups and the value perceived by businesses of creating mashups increased, platform providers entered the ecosystem to fill the void. Initially, these were graphical tools, such as QEDWiki from IBM (http://services.alphaworks.ibm.com/graduated/qedwiki.html), to simplify the composition of APIs into mashups. Platform providers also quickly started to offer marketplaces for APIs and mashups.

At present, there is no leading platform provider, nor a leading marketplace that could serve all user needs. We examined the increase in the complexity of mashups, measured as the average number of APIs combined in a mashup, and plotted it against the timing of the introduction of mashup platforms.

The first set of platforms included libraries, such as the Yahoo! User Interface Library (YUI, http://developer.yahoo.com/yui/), and templates such as those provided by the Ning social networking site (http://www.ning.com/). Later, the first hosted sites for mashups were introduced, such as Coghead (http://www.coghead.com/). The first platform that can be considered a mashup composer, DataMashups.com, was also released around that time, as was the first platform, Dapper (http://www.dapper.net/), for extracting implicit APIs from web sites. Almost two years after the publication of the first mashup, there was a flurry of releases of mashup composers, including the now defunct Teglo and QEDwiki as well as Yahoo! Pipes (http://pipes.yahoo.com/). Many of these composers also integrated interfaces to search for known APIs and to integrate them into a mashup.

We find that mashup platforms have increased in sophistication, from early hosting for mashups and screen scraping tools to more recent graphical mashup composers, in response to the increasing complexity of mashups and the needs of enterprise users. One of the major shifts has been in the types of mashups created: from one-feature mashups to mashups that combine multiple open APIs and internal data sources. The latter type of application requires more advanced tools.

Managerial Implications

The managerial implications encompass three areas related to the creation of mashups and the development of open APIs: i) selection of APIs; ii) introduction of new APIs; and iii) composition of APIs into mashups.

First, our research suggests that the position of a data provider in the mashup ecosystem affects the likelihood of their API being incorporated into a mashup. The number of mashups using a given API is a first indicator of how likely an API will be selected as the basis of a new mashup. For users, the popularity of an API is a signal of its quality. When users select an API, they will give preference to more widely used APIs. But popularity alone does not fully explain the how APIs are selected, except where a mashup consists of exactly one API. In all other cases, the number of interactions with other APIs also plays into the decision to select a given API.

The frequency with which APIs are combined in a mashup is an indicator of how likely they will be combined in future mashups. We observed that the positions of data providers in the mashup ecosystem are mutually reinforcing. One factor we would like to offer as an explanation is that when APIs are used together, significant experience on how to integrate these APIs is obtained. This, in turn, leads developers to prefer proven combinations of APIs when developing new mashups. Another likely factor is that mashups, as the literature on the role of imitation in innovation leads us to conclude (http://mackcenter.wharton.upenn.edu/Research%20Papers/ethiraj4.pdf), are developed by emulating existing mashups. In our research to date, we have not studied the impact of copying or cloning mashups on the mashup ecosystem.

This has implications for users of mashups and data providers. Users will select APIs based on how many other mashups use a given API, as well as the collective experience in using a given API with other APIs to be selected for the mashup. Data providers, when introducing a new API, will benefit from ensuring that their API integrates well with existing APIs that are strongly positioned in the mashup ecosystem. Data providers should look for opportunities to complement existing APIs. By complementing, the new API will also benefit the provider of the existing API by providing additional contexts of use and increasing its potential share of mashups that use it. In order to identify potential niches to enter, data providers need to gain a good understanding of the structure of the current ecosystem. Mapping the mashup ecosystem offers key insights for introducing your own API or mashup.

Second, our analysis suggests that the complexity of mashups drives the development of mashup platforms. The design of more complex mashups requires more sophisticated platforms. This coincides with the increasing interest in enterprise applications of mashups, which may itself be a major contributor to higher complexity. Platform providers need to introduce tools that help manage this complexity. Complexity introduces challenges in searching for APIs, enforcing design rules during the composition of APIs, and certification of APIs. The selection of APIs turns into a problem of finding the right combination of APIs for a given purpose. Enforcing design rules requires a codification of integration experience so it can, at least partially, be automated by a tool. Finally, APIs need to be certified in terms of meeting quality standards.

Conclusion

We have described an approach to map the structure of the mashup ecosystem and its growth over time. The approach uses visualization to show the relationships between APIs and mashups, and subsequently uses network analysis to obtain key characteristics of the ecosystem and identify significant ecosystem members and their relationships. We also discussed the managerial implications of our analysis for data providers, mashup developers, and developers of tools or platforms for the development of mashups.

Future work includes answering the question what laws underlie the growth of the mashup ecosystem and the mechanisms of the creation of mashups. Due to availability of rich data about its structure and growth, the mashup ecosystem gives us a glimpse at innovation processes. We believe that results from the examination of the mashup ecosystem can also shed light on the nature of innovation and of ecosystems in general.

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

LICENSING OF OPEN APIS

"I'd always rather err on the side of openness. But there's a difference between optimum and maximum openness, and fixing that boundary is a judgment call. The art of leadership is knowing how much information you're going to pass on -- to keep people motivated and to be as honest, as upfront, as you can. But, boy, there really are limits to that."

Warren Bennis
http://en.wikipedia.org/wiki/Warren_Bennis

Two current trends in software development are the open source paradigm and the notion of software as a service. The combination of these has lead to the concept of open APIs and mashups. Since late 2005, there has been a rapid proliferation of applications, referred to as mashups, that combine data and services provided by third parties through open APIs with data sources owned by users. Open APIs give users access to the data or services of an information technology (IT) platform. A well-known example is the Google Maps API (http://code.google.com/apis/maps) which generates maps for a given location, whose output can be combined with other data and services into mashups.

Combining services and data from multiple sources raises several issues related to intellectual rights in mashups. However, the concept of mashups is currently in a nascent stage, and service and data providers often underestimate the relevance of these issues. In this paper we give an overview of open API licensing and provide examples from current open APIs. We then briefly discuss licensing of open APIs.

Objectives of an Open API License

The objectives of an open API license are similar to the objectives of a software license. These objectives can be summarized as follows:

1. To define the extent to which the API can be used without constituting an infringement.

2. To have a remedy against the consumer for complaints which do not constitute an infringement of copyrights.

3. To limit the liability of API providers in case of failure of the API.

Anatomy of an Open API License

In this section we describe the anatomy of an open API license. We do not claim that the given anatomy is complete as it is almost impossible to generalize all the terms of a license. Furthermore, this article is not intended as a substitute for legal advice and we highly recommend service providers and service consumers to obtain appropriate legal counsel to make use of licenses for their open APIs.

1. **Subject:** the subject of the license relates to the definition of the open API being licensed. It defines some related information about the open API and may include a unique identification code for the API, name, and other relevant information.

2. **Scope of Rights:** the following rights in an open API allowing extraction and re-utilization of the whole or a substantial part of services and data:

- usage: indicates a set of methods in which the service and data can be consumed

- reuse: indicates a set of operations in which the service and data, or portions of it, can be re-utilised

- transfer: indicates a set of procedures in which the rights over the service and data can be used

The scope of rights of an open API license reflect what can be done with the open API. For example, the clauses of the Google Maps API terms (http://code.google.com/apis/maps/terms.html) include the following:

"If you develop a Maps API Implementation for use by other users, you must:

(a) display to the users of your Maps API Implementation the link to Google's Terms of Use as presented through the Service or described in the Maps APIs Documentation;

(b) explicitly state in your Maps API Implementation's terms of use that, by using your Maps API Implementation, your users are agreeing to be bound by Google's Terms of Use;"

3. Attribution: copyright law refers to attribution as the requirement to acknowledge or credit the author of a work which is used or appears in another work. Attribution signifies a decent sign of respect to acknowledge the creator.

Flickr requires the following attribution terms (http://www.flickr.com/services/api/tos/):

"You shall place the following notice prominently on your application: "This product uses the Flickr API but is not endorsed or certified by Flickr.""

4. Non-Commercial use: commercial uses and non-commercial uses are differentiated by Flickr as follows:

"Flickr is committed to free and open access to our APIs for commercial and non-commercial purposes. However, providing the APIs does have real costs for Flickr. For uses of Flickr APIs over a certain rate or for certain types of commercial applications, Flickr reserves the right to charge fees for future use of or access to the Flickr APIs."

5. Information Rights: the rights over the data created or modified by an open API based on the input from consumers is owned by the consumers. However, API providers can transfer such information to third parties.

These are some of the clauses defining information rights provided by the Google Friend Connect APIs (http://code.google.com/apis/friendconnect/terms.html):

"You agree that Google may transfer and disclose to third parties personally identifiable information about you for the purpose of approving and enabling your use of the Services, including to third parties that reside in jurisdictions with less restrictive data laws than your own.

Google may share non-personally-identifiable information about you, including Web site URLs, site-specific statistics, and similar information collected by Google, with advertisers, business partners, sponsors, and other third parties. In addition, you grant Google the right to access, index and cache your Web sites, or any portion thereof, including by automated means including Web spiders or crawlers."

6. Financial Terms: often, open APIs are charged on a pay-per-use or transaction base. This transaction-based model allows API providers to charge for each use, as the license defines the term “use.” The use of the API can be continuously recorded and monitored by service management systems. This model of pricing is quite similar to charging for utilities like electricity and water.

Subscription is an alternative financial model that allows consumers to purchase the open API based services for a fixed term, during which time they automatically receive full support from service providers including any upgrades or feature enhancements.

This is a subset of the clauses of the financial terms of Amazon web services (http://aws.amazon.com/agreement/):

"In consideration of your use of any of the Paid Services, you agree to pay applicable fees for Paid Services in the amounts set forth on the respective Service detail pages on the AWS Website (including any minimum subscription fees). You are responsible for any fees assessed by Amazon Payments for transactions that you submit to the Payment Service using Amazon FPS. Fees for any new Service or new Service feature will be effective upon posting by us on the AWS Website for the applicable Service."

7. Warranty: in general, an open API is licensed by the licensor "as is" and without any warranty of any kind, either express or implied, whether of title, of accuracy, of the presence or absence of errors, of fitness for purpose, or otherwise.

These are the some of the warranty clauses of the Google Maps API (http://code.google.com/apis/maps/terms.html):

"YOU EXPRESSLY UNDERSTAND AND AGREE THAT YOUR USE OF THE SERVICE AND THE CONTENT IS AT YOUR SOLE RISK AND THAT THE SERVICE AND THE CONTENT ARE PROVIDED "AS IS" AND "AS AVAILABLE."

ANY CONTENT OBTAINED THROUGH THE USE OF THE GOOGLE SERVICES IS DONE AT YOUR OWN DISCRETION AND RISK AND YOU WILL BE SOLELY RESPONSIBLE FOR ANY DAMAGE TO YOUR COMPUTER SYSTEM OR OTHER DEVICE, LOSS OF DATA, OR ANY OTHER DAMAGE OR INJURY THAT RESULTS FROM THE DOWNLOAD OR USE OF ANY SUCH CONTENT."

8. Limitation of Liability: limitation of liability clauses limit the liability of the licensor and the licensee under the license agreement. Under this clause, both parties disclaim liability for unforeseeable damages, such as network errors or hosting server problems, or indirect damages. Limitation of liability clauses often include a ceiling for monetary liability.

The limitation of liability clause of Flickr is as follows (http://www.flickr.com/services/api/tos/):

"FLICKR SHALL NOT, UNDER ANY CIRCUMSTANCES, BE LIABLE TO YOU FOR ANY INDIRECT, INCIDENTAL, CONSEQUENTIAL, SPECIAL OR EXEMPLARY DAMAGES ARISING OUT OF OR IN CONNECTION WITH USE OF THE FLICKR APIS, WHETHER BASED ON BREACH OF CONTRACT, BREACH OF WARRANTY, TORT (INCLUDING NEGLIGENCE, PRODUCT LIABILITY OR OTHERWISE), OR ANY OTHER PECUNIARY LOSS, WHETHER OR NOT FLICKR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. UNDER NO CIRCUMSTANCES SHALL FLICKR BE LIABLE TO YOU FOR ANY AMOUNT."

Conclusion and Future

Licenses for open APIs reflect the differences between traditional software and web services as they govern the execution, reuse and composition of services and data exposed by third-party systems. It is common for API providers to offer some of their APIs for free and others for pay. A provider can even use separate terms for the same API by offering different licenses for high-end and low-end versions of the service provided by the API.

The common ways for a provider to impose constraints on the execution of a service include limiting requests, results, and quality.

In the future, we believe that more open APIs will begin to be licensed using open source licenses. While the source code of the interface of an open API is always available, open sourcing an open API makes the source code of the API implementation available in addition to the source of its interface. In this case, users would be able to modify an API, or derive new APIs from an open API. However, in order to avoid license forking, providers would like to prevent the new open API from being licensed differently from the parent API. In this way, the value added by the changes can be returned to the community of users of the parent API.

At present, very few open APIs fit the description of being open sourced. One example is WikiDot (http://wikidot.com), a wiki service with an open API that is licensed under the GNU Affero General Public License (http://www.fsf.org/licensing/licenses/agpl-3.0.html). We expect more open APIs to be available under an open source license in the near future.

Dr. G.R. Gangadharan is a research scientist at the Novay (Telematica Institute), Enschede, The Netherlands. His research interests are mainly located on the interface between the technological perspective and the business perspective. His research interests include Service Oriented Computing, Internet Software Engineering, Intellectual Property Rights, Free and Open Source Systems, and Business Models for Software and Services.

Recommended Reading

Analyzing Software Licenses in Open Architecture Software Systems
http://www.ics.uci.edu/~wscacchi/Papers/New/ICSE2009-FLOSS-Workshop.pdf

Mashup Arts Licensing Terms
http://www.mashuparts.com/licensing/

At Mashup Camp, Geeks Plot Future of Web
http://news.zdnet.com/2100-3513_22-150920.html

"A rich Internet application combines the benefits of using the Web as a low-cost deployment model with a rich user experience that's at least as good as today's desktop applications. And, since RIAs don't require that the entire page be refreshed to update their data, the response time is much faster and the network load much lower. Think of a globally available client/server application."
http://flexblog.faratasystems.com/?p=163

Since 2004, the number of Rich Internet Applications (RIAs, http://en.wikipedia.org/wiki/Rich_Internet_application) has increased, making them a key component of the Web 2.0 phenomenon. Many RIAs have been developed using JavaScript (JS) and AJAX (Asynchronous JavaScript And XML). AJAX is used to access remote data sources, that reside on the server or are available through an open API, directly from within the application. The rich variety of applications would not be available today without the concurrent appearance of many powerful JavaScript toolkits that have taken the development of these applications from labour-intensive to nearly painless. These toolkits provide an open source alternative to the proprietary products developed by Adobe (Flash/Flex/Air) and Microsoft (Silverlight).

This article provides two examples that demonstrate the evolution of RIAs, then compares the features of the most commonly used JavaScript Toolkits used to create RIAs. We then discuss how freely available toolkits are able to compete against proprietary alternatives. Finally, we provide some concluding remarks based on our experience with creating enterprise RIAs.

Evolution of RIAs

We have been involved in the rise of RIAs right from the beginning. At digg.com, we adopted AJAX before the acronym was created, when it was basically referred to by the much less friendly acronym XMLHTTP. One of the early features that drove a lot of traffic to digg was "Digg Spy". Originally developed in February 2005 with JavaScript, it was greatly enhanced in July 2005 via the use of a couple of the first JavaScript toolkits to appear: Prototype (http://prototypejs.org) and Scriptaculous (http://en.wikipedia.org/wiki/Scriptaculous). Digg Spy provided a near real time view of activity on the site without resorting to the meta-refresh technique in common use at the time. What was innovative then can now be seen all over the Web, most noticeably at sports sites such as espn.com and nfl.com. These sites were the largest users of the clunky meta-refresh approach, and have since heavily invested in AJAX.

Several years later at tripadvisor.com, I was deeply involved in the development of TripAdvisor Flights, a meta search product for travel planning with a huge interactive front-end component, developed using jQuery (http://en.wikipedia.org/wiki/Jquery) and Prototype. The evolution of JavaScript toolkits over the past four years has been astounding, and can be seen clearly when contrasting these two products. It's interesting to note that the original Digg Spy was retired recently in favour of the proprietary Adobe Flash product, and that had more to do with an Adobe sponsorship of the product than technical merit.

JavaScript Toolkits

The sheer number of JS toolkits can be confusing to the uninitiated. In addition to technical considerations, issues such as licenses, modularity, and support should be considered.

We discuss the most common toolkits: Prototype/Scriptaculous, Dojo, YUI, Ext-JS and jQuery.

Every toolkit, either in its core or with closely associated packages, provides support for a variety of things inherited from desktop applications that are not easy to do in JavaScript and which are especially difficult to do in a cross-browser compatible fashion. Drag-n-drop, tooltips, status bars, windows, modal dialogs, and progress bars are present in all of the toolkits. Additionally, most provide a set of widgets like tree controls, combo boxes, sliders, rich text editors and other controls inherited from their desktop predecessors. It's not entirely clear that these controls actually make people more comfortable on the Web. Our experience has been that most people have adapted to the dumbed-down interface of the web browser and are often confused by additional controls.

One important, but not obvious, consideration is that JavaScript does not support the class-based inheritance model (http://en.wikipedia.org/wiki/Class-based_programming) that is familiar to most software developers. It instead uses a "prototype" based inheritance. Many of the toolkits provide a mechanism for class-based inheritance as a way to make development more attractive.

1. **Prototype:** Prototype is the oldest framework, and thanks to its close association with Ruby on Rails, it is a widely adopted rapid application development (RAD, http://en.wikipedia.org/wiki/Rapid_application_development) framework for the web. Developed by Sam Stephenson in 2004, it's probably the best-developed class-based system. Scriptaculous is a separate library, built on Prototype, that provides effects and animation. Scriptaculous has gained a reputation for being bloated, and that caused Digg to abandon both Scriptaculous and Prototype in favor of jQuery.

2. **Dojo:** Dojo (http://en.wikipedia.org/wiki/Dojo_Toolkit), unlike the rest of the toolkits mentioned, provides a rich text editor, something for which there is a great demand. It has also completed a number of partnerships, most notably with Zend. Zend is a leading framework for PHP development, supported by IBM and SUN, and they plan to integrate Dojo in their next version. In addition to the usual collection of widgets, Dojo provides extensions to do box and line charts.

3. **YUI:** YUI (http://en.wikipedia.org/wiki/Yahoo!_UI_Library) is the kitchen sink of JS toolkits, chock full of widgets and gadgets, with Yahoo as its corporate backer. There is a much smaller core available, countering any accusation of bloatedness. Overall, it has the richest set of widgets, and the big company support is a decisive factor for many websites. YUI also includes a useful JavaScript compressor that we use at tripadvisor.com to decrease download times and to save bandwidth.

4. **Ext-JS:** Ext-JS (http://en.wikipedia.org/wiki/ExtJS) has a dual commercial/GPL license which has spurred some licensing issues that have impaired its adoption. Most JS toolkits use the LGPL or MIT license, which has allowed them to be used without developers having to open source their server side code. Ext-JS provides a rich set of widgets, and provides integration with Google Web Toolkit (http://code.google.com/webtoolkit/), a server-side framework that generates JavaScript.

5. **jQuery:** jQuery (http://en.wikipedia.org/wiki/Jquery) has the usual widgets available under the rubric jQuery UI (http://jqueryui.com). It's probably best known for its terseness, a result of a technique called "chaining" that makes for very concise code. This is important for JavaScript which almost always has to be transferred from a server to a client machine over the Internet. Its greatest strength, chaining, is also the main point of criticism as jQuery code looks completely different from JavaScript code. It has benefited recently from a close association with the Mozilla Foundation; John Resig, the initial developer, is employed there.

Competing with Commercial Alternatives

One of the interesting dynamics at play in the Internet industry and the continuing evolution of RIAs is the ever present rhetoric pushing the latest proprietary alternatives. The toolkits described above have been instrumental in ensuring that Java Script can compete, even in a world with many different browsers, each with their own quirks and bugs. Ten years ago it would have been hard to argue that the number of browsers in use would increase. Now, it's a given, with Opera, Chrome, and Safari all holding their own against Firefox and Internet Explorer. This has presented challenges for Java Script developers and a lot of great work by countless individuals has ensured that the open source alternatives continue to be preeminent.

The canonical cool application for Java Script is still Gmail, and its success has demonstrated that useful and usable applications can be done in pure JS. Other well known applications include Zimbra (http://en.wikipedia.org/wiki/Zimbra) and Zoho Applications (http://en.wikipedia.org/wiki/Zoho). Beyond the obvious rivalry of JavaScript vs. Flash, the continuing success of JavaScript in developing RIAs also represents another success for open source. As the adoption of browser-based applications increases, the reliance on many different types of proprietary desktop applications decreases. This is good news since the adoption of open source browsers continues to be on the rise.

Concluding Remarks

One possible concern with the use of JavaScript, Ajax and rich JavaScript applications is the fact that the original XMLHTTP protocol is itself a proprietary protocol owned by Microsoft which was originally released as an ActiveX object. Other vendors have implemented their own versions of the protocol and there are other ways to do the dynamic updates that XMLHTTP provides, should Microsoft ever decide to be less benign about the protocol.

JavaScript has a clear lead in the next big frontier in applications, that of mobile development. In particular, the iPhone doesn't support any of the proprietary alternatives, but it does support JavaScript. Most applications, including our own JavaScript heavy flights product, work quite well in the iPhone version of Safari. Flash and Silverlight don't, at least not yet.

Our recent meta search application turned out to be quite large and challenging, but the use of JavaScript and jQuery turned out to be a key contributor to its apparent success.

It still took a great deal of effort to make it perform adequately while working across all major browsers. As most web developers know, the weak point for these applications continues to be the large installed base of Internet Explorer 6.0. The DOM and JavaScript implementation in that browser has a huge number of bugs, and requires many workarounds and often the foregoing of best practices. We can only hope that the recent launch of Internet Explorer 8.0 will prompt at least some people to upgrade.

Owen Byrne is currently Senior Manager of Travelpod Labs. He is probably best known as the co-founder and original developer of digg.com where he was the primary technical decision maker for most of its period of growth, from inception to the Series A financing. Owen holds three degrees from Saint Mary's University and Dalhousie University, as well as an ABD from the University of Manitoba. He has over 20 years experience in software development and managerial roles including a brief stint as a university professor.

Recommended Resources

RIApedia
http://www.riapedia.com/

The Business Benefits of Rich Internet Applications for Enterprises
http://www.ashorten.com/wp-content/uploads/2009/01/Adobe_RIA_Enterprise_Web0109.pdf

Presentations from the Business of APIs Conference
http://www.apiconference.com/presentations/

How To Roll Out An Open API
http://radar.oreilly.com/2005/05/how-to-roll-out-an-open-api.html

How do I monetize an API?
http://blog.mashery.com/2008/06/18/how-do-i-monetize-an-api/

MEASURING MODULARITY IN OPEN SOURCE CODE BASES

"... the act of properly designing a complex system characterized by a modular architecture is not a trivial task. On the contrary, modularity bears high costs: careful modularization is a cognitive challenging activity, since it translates in devising a decomposition of the whole system in autonomous subparts whose interdependencies are reduced to the minimum. Moreover, failure to perfectly modularize an architecture results in costs of dealing with fine tuning and tempering activities aimed at solving unexpected and unforeseen interdependencies."

http://econpapers.repec.org/paper/trtrockwp/020.htm

Modularity of an open source software (OSS) code base has been associated with growth of the software development community, the incentives for voluntary code contribution, and a reduction in the number of users who take code without contributing back to the community. As a theoretical construct, modularity links OSS to other domains of research, including organization theory, the economics of industry structure, and new product development. However, measuring the modularity of an OSS design has proven difficult, especially for large and complex systems.

In this article, we describe some preliminary results of recent research at Carleton University that examines the evolving modularity of large-scale software systems. We describe a measurement method and a new modularity metric for comparing code bases of different size, introduce an open source toolkit that implements this method and metric, and provide an analysis of the evolution of the Apache Tomcat application server as an illustrative example of the insights gained from this approach. Although these results are preliminary, they open the door to further cross-discipline research that quantitatively links the concerns of business managers, entrepreneurs, policy-makers, and open source software developers.

Modularity and Design Structure Matrices

In "Design Rules: The Power of Modularity" (http://mitpress.mit.edu/catalog/item/default.asp?tid=3606&ttype=2), Baldwin and Clark present a major study of the role of modularity in the evolution of the computer industry. They argue that modularity is an important and fundamental connection between at least three different systems: i) the engineering design; ii) the organization of the people who implement and maintain the design; and iii) the network of companies forming the economic system around the design. Each system constrains and enables the other two. Other researchers have since built on and extended this work, which in its original form did not directly address OSS.

According to Baldwin and Clark, a module is a unit whose structural elements are strongly connected among each other and relatively weakly connected to elements in other units. Just as there are degrees of connectedness, there are degrees of modularity. This motivates the interest in metrics and techniques to measure the modularity of the structures comprising an artifact. In a widely-cited 2006 paper titled "The Architecture of Participation: Does Code Architecture Mitigate Free Riding in the Open Source Development Model?" (http://www.people.hbs.edu/cbaldwin/DR2/BaldwinArchPartAll.pdf), Baldwin and Clark provide a theoretical argument that a more modular open source code base will attract more voluntary contributions and have less free riding of non-contributors than one that is less modular.

When the complexity of one of the elements crosses a certain threshold, that complexity can be isolated by defining a separate abstraction that has a simple interface. This abstraction hides the complexity of the element and the interface indicates how the element interacts with the larger system. Modularity decreases complexity in several ways. In particular, it allows designers to focus on individual modules rather than the whole integrated artifact. This radically changes the design process and allows for work on individual modules to be parallelized.

The Design Structure Matrix (DSM) is an analysis tool for mapping complex systems. It provides a compact representation of a complex system that visualizes the interdependencies between system elements. According to Baldwin and Clark, “it is a powerful analytic device, because by using it we can see with clarity how the physical and logical structure of an artifact gets transmitted to its design process, and from there to the organization of individuals who will carry the process forward.”

A DSM is a square matrix with off-diagonal cells indicating dependencies between the system elements. A value in the cell at row i and column j means that the element at position i depends in some way on the element at position j. For example, the design elements could be Java classes and dependencies would be references between classes. This information can be extracted automatically from the code base. Clustering reorganizes the DSM elements to more clearly visualize and analyze dependency relationships.

The details of how this analysis is conducted are quite technical, and it builds on prior research in several related domains. Alan MacCormack, John Rusnak, and other colleagues at Harvard Business School recently published two important advances.

In a 2006 article titled "Exploring the Structure of Complex Software Designs: An Empirical Study of Open Source and Proprietary Code" (http://opensource.mit.edu/papers/maccormackrusnakbaldwin.pdf), they employed DSMs to empirically compare the design structures of two software products: the Linux kernel and the Mozilla web browser. They proposed a clustering algorithm to measure dependencies. However, their comparison critically depended on selecting versions of the systems with a similar number of source files which were the elements in the DSM. One motivation of our work was to remove this restriction, and to allow the comparison of code bases of different size.

A follow-on article in 2008 examines the evolution over time of two software products: the open source Apache Tomcat application server and an unnamed closed source commercial server product. A coarse metric is introduced that represents the change ratio between the consecutive versions in the product evolution. The authors conclude that DSMs and design rule theory can explain how real-world modularization activities allow for different rates of evolution to occur in different modules, and create strategic advantage for a firm.

Measuring Design Evolution

Our method for examining the evolving modularity of large-scale software systems implemented in Java builds on the discussed DSM methods and algorithms, but differs from past work in several aspects. As with these approaches, we: i) automate dependency extraction from the software code base; ii) employ design structure matrices for visualization and analysis of dependency information; and

iii) compute cost metrics as measures of modularity. We differ from the earlier work in: i) that our unit of analysis uses Java classes rather than C source files; and ii) our use of the relative clustered cost metric.

Our design elements are Java classes and our dependencies are references between classes, whether by inheritance, declared fields, or method calls. Because dependencies between Java classes can be extracted from the compiled code of a software system, we need only obtain binary distributions of the selected versions. The steps of our method are as follows:

1. Select the versions to be analyzed and obtain their binary distributions.

2. For each version, extract the dependency information from the compiled code.

3. Create DSM instances and extract cost metrics.

We implemented three modularity metrics:

1. **Propagation cost:** measures the extent to which a change in one element impacts other elements. It is a representation of the degree of coupling without consideration of the proximity between elements.

2. **Clustered cost:** a more sophisticated metric that assigns different costs to dependencies based on the locations of elements within clusters. It has an important limitation in that it can only be used to compare DSMs of similar sizes.

3. **Relative clustered cost:** (our contribution) extends the clustered cost metric to compare DSMs of different sizes, avoiding the limitation of the clustered cost metric.

More formal definitions of these metrics and details of their implementations will be seen in our upcoming presentation "Design Evolution of an Open Source Project Using an Improved Modularity Metric" (http://oss2009.org/index.php?id=preliminary_program.htm).

Evolving Modularity of Tomcat

We used our method to study the evolving code base of an open source system, the Apache Tomcat application server developed and maintained by the Apache Software Foundation (http://apache.org). Tomcat is implemented in Java, and has two major distinct functional modules: the Tomcat-main server core and Jasper, a separate module that processes Java Server Pages. Tomcat-main and Jasper are linked only through the J2EE API (http://en.wikipedia.org/wiki/J2ee).

Over the ten-year period between 1999 and 2008, four major versions of Apache Tomcat were released:

• Apache Tomcat 3.x is based on the original implementations of the Servlet 2.2 and JSP 1.1 specifications donated by Sun Microsystems

• Apache Tomcat 4.x implements the Servlet 2.3 and JSP 1.2 specifications and Catalina, a new servlet container based on a different architecture

• Apache Tomcat 5.x implements the Servlet 2.4 and JSP 2.0 specifications

• Apache Tomcat 6.x implements the Servlet 2.5 and JSP 2.1 specifications

Since support for specific standards specifications is of primary importance to Tomcat users, major version numbers for Tomcat mirror the versions of the Servlet and JSP specifications that Tomcat supports.

However, a change in major version numbers does not necessarily correspond to major changes in the structure of the code base. Thus, when we selected the versions of Tomcat for our analysis, we identified significant architectural events in the evolution of the Tomcat code base, such as major changes to the architecture to improve performance, or the introduction of the Catalina servlet container.

For each version, we examined the Tomcat-main and Jasper modules both separately and in combination. For each analysis, we computed the number of classes, number of dependencies, propagation cost, number of vertical busses, number of clusters, clustered cost, and relative clustered cost.

The number of classes nearly tripled between version 3.0 and 6.0.16. This is clear evidence of the need for modularity measures that permit comparisons of code bases of different size.

Initially, we expected the modularity of Tomcat to increase throughout the evolution of the product. The rationale for this expectation was that as a system evolves, its structure would be continually examined by developers. Specifically, we expected that architectural improvements would also lead to increased modularity. For example, when Tomcat 4.x introduced a new implementation of the servlet container based on a new architecture (Catalina), we expected the new architecture to be more modular because it was built from the ground up for flexibility and performance.

Figure 1: Propagation Cost of Tomcat-main

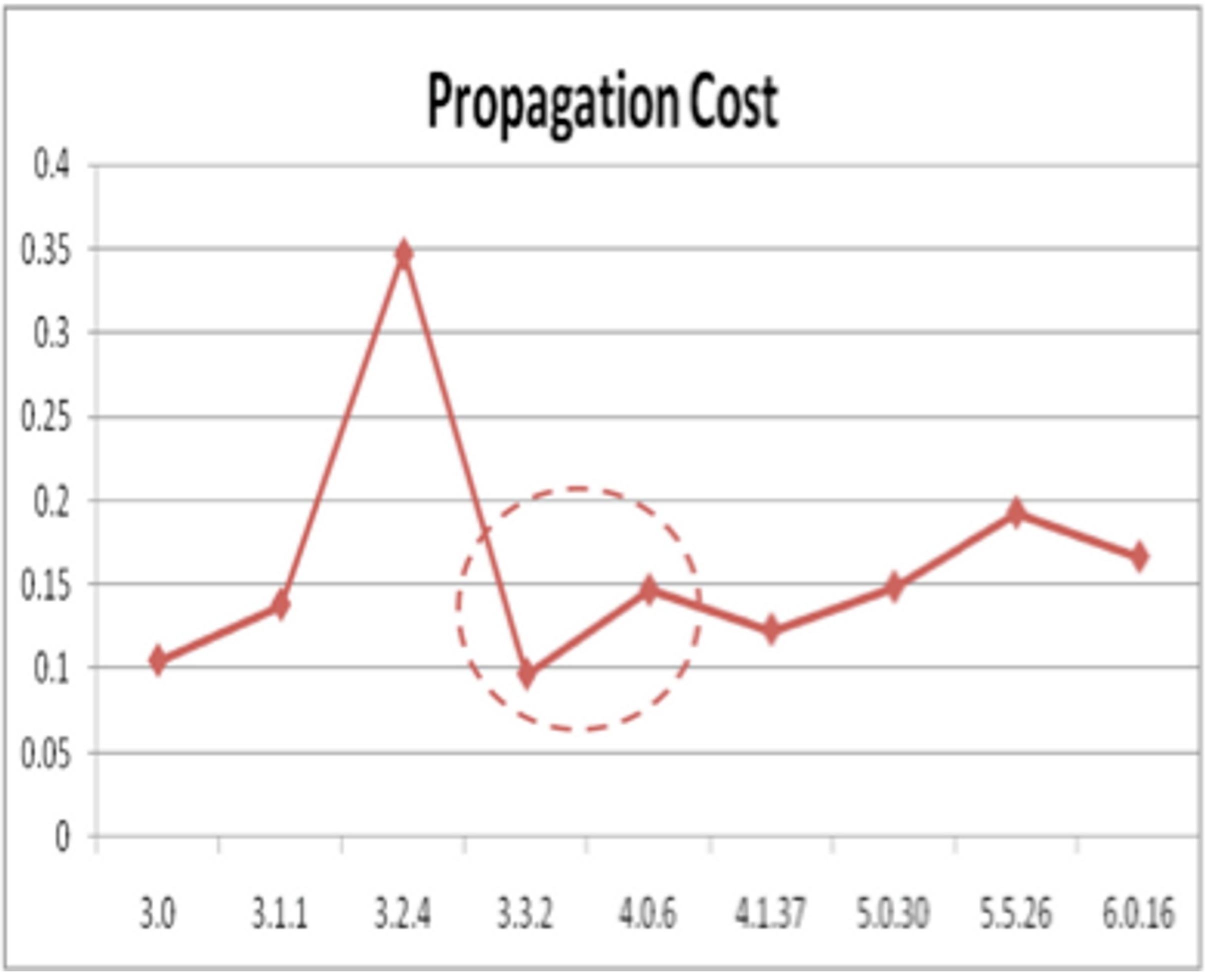
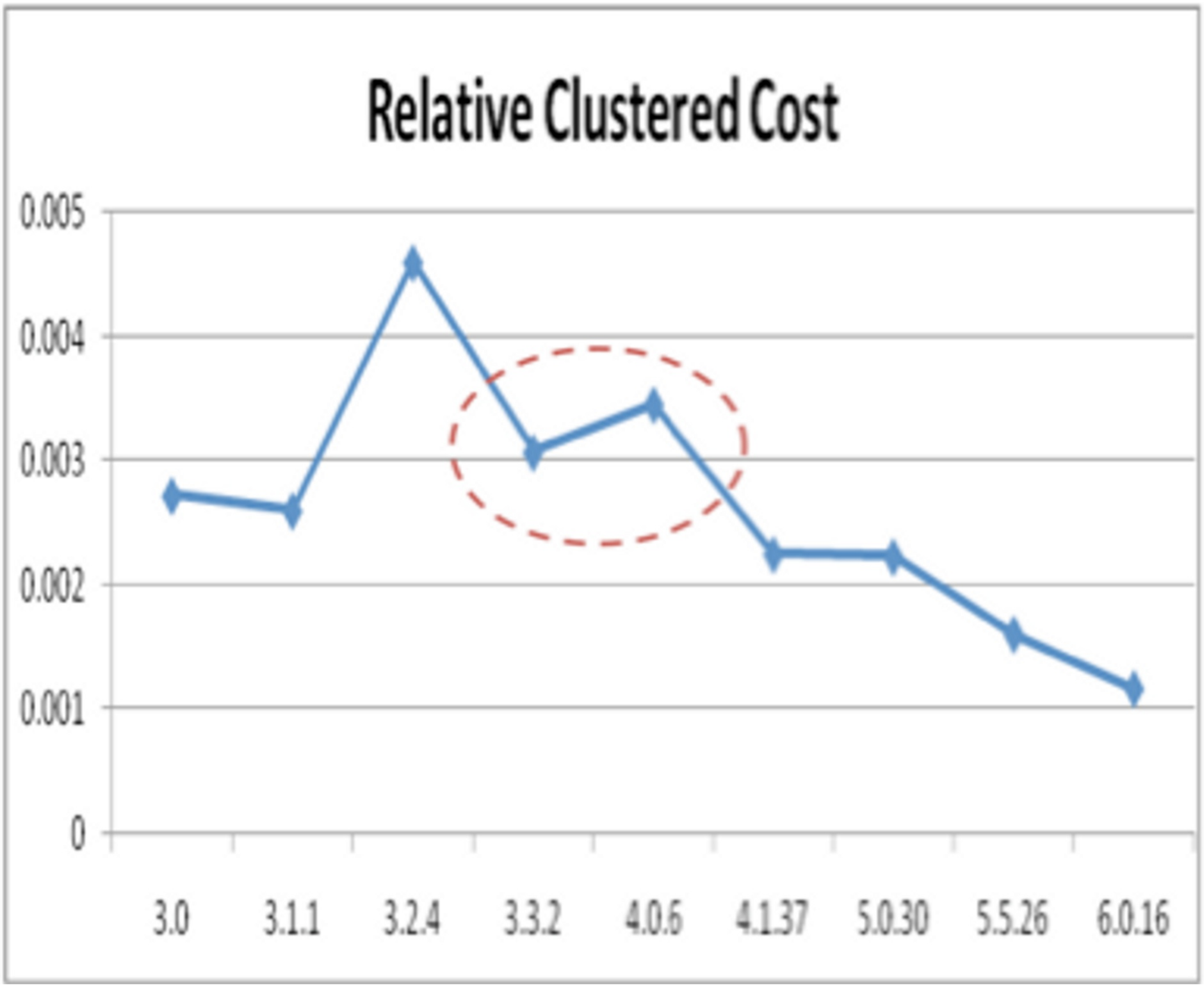


Figure 2: Relative Clustered Cost of Tomcat-main



However, as seen in Figures 1 and 2, we observe that the propagation costs for Tomcat 3.3.2 and 4.0.6 are 9.6% and 14.6%, respectively, and the relative clustered costs are 0.0031 and 0.0035. Both metrics suggest that version 4.0.6 is less modular than version 3.3.2, the opposite of what we expected to find. Version 3.3.2 is the latest production release of Tomcat 3.x which finished the refactoring effort and introduced a more modular design by allowing the addition and removal of modules that control the execution of servlet requests. Version 4.0.6 is the final release of Tomcat 4.x that introduced the Catalina servlet container. A similar pattern occurred when major architectural changes were made to the Jasper subsystem at other points in time.

A closer examination of the events surrounding these spikes in propagation cost and relative clustered cost suggests that each decrease in modularity was precipitated by a major architectural or implementation change. For all other releases, whether major versions or incremental releases, the code became increasingly more modular. Interestingly, each spike is immediately followed by an increase in modularity. In fact, in each case, the increase in modularity of the consecutive version more than compensated for the previous decrease.

Our data is not conclusive on why this pattern occurred, but we offer a plausible explanation. Once new functionality is initially deployed and working, focus shifts. Developers revisit the design and perform refactoring and cleanup activities which represent changes to the structure of the system, but not to its behaviour. Increased understanding and experience gained through the original implementation permits developers to more easily restructure the existing code into a more modular design. The result is a significant increase in modularity that compensates for the original decrease in the previous version.

To capture these observations, we propose three propositions that can guide future research on the evolution of modularity of software systems:

Proposition 1: major architectural and implementation changes cause the modularity of a software system to decrease at first.

Proposition 2: major changes are followed by periods of refactoring and cleanup activities, which cause the modularity of the software system to increase again.

Proposition 3: the increase in modularity as a result of refactoring and cleanup activities more than offsets the decrease in modularity due to a major change.

Conclusion

This paper reported on recent advances towards understanding the evolution of large OSS systems, and proposed an improved modularity metric based on DSMs that allows the comparison of code bases of different size. Our research provides initial evidence that as a large software system evolves, major architectural changes, at first, lead to an increase in modularity, but are followed by refactorings and cleanup activities which lead to a subsequent increase in modularity.

Although these results are preliminary, they are part of the larger research program in which we hope to provide deeper insights into the connections between technical, organizational, and economic systems.

Steven Muegge is a faculty member of the Department of Systems and Computer Engineering at Carleton University, Ottawa, Canada. Professor Muegge teaches within the Technology Innovation Management program. His current research interests include open source software, open innovation, and open source ecosystems.

Roberto Milev completed an M.Eng. degree in Technology Innovation Management in 2008. As part of his research into open source software, he derived the relative clustered cost metric and developed the jDSM open source toolset for computing DSMs and modularity metrics. He is currently working as a manager for a software development company.

Recommended Resources

Java DSM library
http://jdsm.sourceforge.net/

DSM Home Page
http://www.dsmweb.org/

"Open source software relies upon copyright law: both its protections, and exceptions."

David Fewer, CIPPIC.ca

The University of Ottawa, Faculty of Law is Canada's premiere legal program in law and technology. The Torys Technology Law Speaker Series (http://web5.uottawa.ca/techlaw/en/events/torys-technology-law-speaker-series/) brings prominent speakers from around the world to discuss current topics in law and technology.

A new approach to open source software (OSS) was presented to students and faculty at the University of Ottawa on March 11, 2009. Michael Madison, Associate Dean for Research and Professor of Law at the University of Pittsburgh School of Law, presented "Open Source Licenses and the Boundaries of Knowledge Production". Prof. Madison spent time outlining and answering questions on a novel interpretation of copyright in the age of OSS. Using historical examples, he called for the courts to incorporate a "spatial framework" to deal with open source licenses. His approach was particularly relevant and timely in light of a recent opinion from the US Court of Appeals for the Federal Circuit, Jacobsen v. Katzer (http://en.wikipedia.org/wiki/Jacobsen_v._Katzer).

Lessons Learned from Jacobsen v. Katzer

The presentation began with a detailed case history in the fight between Robert Jacobsen, manager of an OSS project hosted on SourceForge called the Java Model Railroad Interface (JMRI, http://sourceforge.net/projects/jmri/), and Matthew Katzer and Kamind Associates, Inc. who collectively develop commercial software products for the model train industry. Prof. Jacobsen, among other things, sought a declaration from the courts recognizing that Mr. Katzer's use of the JRMI software was in violation of the open source license offered by the project. A preliminary injunction was then sought under copyright infringement. The appellate court decision held that the Artistic License (http://opensource.org/licenses/artistic-license-2.0.php) granted by the JMRI to users of the model train software, and, more generally, OSS licenses, are enforceable. These licenses are set conditions, rather than merely covenants, regarding the use of the copyrighted work. As conditions, a breach of the license may terminate the contract, allowing the copyright holder to sue for infringement.

Although the Court of Appeal ultimately remanded the case back to the district court, two important principles were set forth in the judgment. First, the decision recognized that work can still be protected by copyright even when given away for free. A breach in the conditions in the license makes a user susceptible to copyright infringement as he or she can no longer rely on the license against such claims. Secondly, the court took a purposive approach in legitimizing the nature of the open source collective. It recognized that the OSS movement's growth flowed from the sense of community and credit one gets through contribution and not from common notions of monetary gain. According to the court, "[t]hese restrictions were both clear and necessary to accomplish the objectives of the open source licensing collaboration."

In drawing attention to Jacobsen, Prof. Madison noted that the court's opinion was neither novel nor new. He used a number of historical examples to illustrate how courts have been inconsistent, though tending towards not upholding restraints, in deciding whether "the covenant runs with the code." He used half a dozen examples where "equitable servitudes on chattel" strive to create an obligation to either do something or refrain from doing something through simple possession of the object. These equitable servitudes try to go beyond mere contract law by stating that the property right in an object is constrained by the obligation attached. In the case of Jacobsen, a condition attached by the Artistic license was attribution to the JRMI. According to the examples cited in the US, a 'single use' stamp on a cotton tie was deemed enforceable, but a stamp of 'not for resale' on a record/CD/blue-ray disc was not (http://madisonian.net/2008/04/30/equitable-servitudes-in-packaging and http://en.wikipedia.org/wiki/Post-sale_restraint).

In bringing up these examples, Prof. Madison sought to highlight the special, and often unpredictable, arena where OSS exists. There is no single universally accepted criterion for when a covenant becomes a condition. Sometimes it is based on patent law, other times on copyright, customary use or practice. This incongruity and confusion in determining an object's status is exposed in the alternate decisions from the trial and appeal court in Jacobsen. In remanding the case and sending it back to the lower court, the appeal court recognized that a condition such as the attribution guarantee in the Artistic License was enforceable on the license holder.

Licensing in a Spatial Framework

Prof. Madison believes it is time for the courts to recognize a new framework for this type of litigation. Instead of using a linear temporal approach to decide if the equitable servitude meets certain criteria or fits within a certain box, a court should base its decision on the purpose of the introduced limitation. Instead of relying on the structure or trying to massage the servitude into an 'if-then narrative', Prof. Madison believes the court should take a more abstract approach in applying a spatial framework that looks to the purpose of the condition or license.

OSS relies on copyright and the conditions set forth in the licensing agreement are necessary to uphold the open source paradigm. In order for users to contribute code and not incorporate openly accessible software into their own projects free of payment or attribution, OSS projects must turn to copyright law. Although the conditions may not be economic, they are vital in order to encourage and maintain the integrity of the social alliance.

While open source licenses can fit into the well-known temporal framework, Prof. Madison suggests that recognizing a spatial framework is better suited. As recognized by the appellate court in Jacobsen, the license utilized in open source projects enables open source users to come together from all over the world. These communities utilize the license to encourage different models of creation as well as different types of consumption, reuse, and development. Open source flourishes upon these ideals.

Prof. Madison states that a spatial approach in recognizing the importance of the open source licenses better reflects the community's goals. Such an approach allows open source users to have better recourse against infringers. Potential remedies would also be affected. Normally, courts are very reluctant to grant noneconomic relief. Although it may create problems in other areas, Prof. Madison believes a spatial framework would find courts more susceptible to injunctions that recognize the non-economic importance of open source licenses.

Finally, Prof. Madison discussed the spatial model finding more traction in open source licensing when acting as the proverbial shield rather than sword. He cited the GPLv3 example of license termination when a user applies to patent a project incorporating open source code. In such situations, the GPLv3 explicitly terminates the user's license and makes him or her liable for copyright infringement. Such a defensive strategy protects the open source community from claims of patent infringement and explicitly uses the spatial metaphor to set up a protective zone around the open source project.

Conclusion

Prof. Madison's approach is still being developed and increased discourse and reflection will invariably occur as open source licenses are further utilized and tested by courts. Whether or not a spatial model takes hold in a post-Jacobsen world is unknown, but open source users should take heart that the US appellate court upheld the enforceability of OSS licenses and the importance of copyright to building OSS communities.

Byron Thom is finishing his law degree at the University of Ottawa, Faculty of Law with a concentration in law and technology. His interests vary from new approaches to intellectual property law to how technology may save the world from global warming. Byron was also a participant in Canada's first class on the Law of Robotics and was at the table when Kerr's Postulate was formed.

Recommended Resources

Michael Madison's Law and Tech Blog
http://madisonian.net/

Law and Technology Program at the University of Ottawa
http://www.commonlaw.uottawa.ca/tech

Video of Lecture
http://www.fosslc.org/drupal/node/320

May 4-6

MCeTech

Ottawa, ON

The 4th International MCETECH Conference on e-Technologies aims at bringing together researchers, decision makers, and practitioners interested in exploring the many facets of Internet applications and technologies.

http://www.mcetech.org/

May 6-9

Libre Graphics

Montreal, QC

LGM 2009 is the fourth annual worldwide meeting of teams developing open source graphics applications. Designers, graphic artists and anyone involved in print production and/or web development are cordially invited to attend and meet the developers one to one.

http://libregraphicsmeeting.org/2009/

May 8

WordCamp

Toronto, ON

WordCamp is a conference type of event that focuses squarely on everything WordPress. Everyone from casual end users all the way up to core developers show up to these events. These events are usually highlighted by speeches or keynotes by various people.

http://phug.ca/wordcamptoronto

May 8-9

BSDCan

Ottawa, ON

BSDCan has established itself as the technical conference for people working on and with BSD based operating systems and related projects. The organizers have found a formula that appeals to a wide range of people from extreme novices to advanced developers.

http://www.bsdcan.org

May 10-13

CNIE International Conference

Ottawa, ON

With an expected attendance of over 400 national and international delegates working in the fields of educational technology, health education, K-12 education, multi-media design and distance learning, the 2009 CNIE International Conference offers a unique opportunity for learning, networking and idea exchange. Join colleagues from across the education spectrum discussing, debating and exploring the integration of learning and technology.

http://www.learningconference.ca/cnie2009

May 13-15

SummerCamp

Ottawa, ON

This event will bring together industry, academia, government, and community to learn about open source and to encourage cross pollination of ideas and talent.

http://www.fosslc.org/drupal/summer camp2009

May 16-17

MSR Mining Challenge

Vancouver, BC

The MSR Mining Challenge brings together researchers and practitioners interested in applying, comparing, and challenging their mining tools and approaches on software repositories for open source projects. This year's challenge examines the GNOME Desktop Suite of projects and how they interact.

http://msr.uwaterloo.ca/msr2009/challenge/index.html

May 16-24

ICSE

Vancouver, BC

ICSE provides a forum for researchers, practitioners and educators to present and discuss the most recent innovations, trends, experiences and concerns in the field of software engineering.

http://www.cs.uoregon.edu/events/icse2009/home/

May 17-22

Open Source Programs for Mac

BC Public School System

In this knowWEEK we will look at open source programs for Mac that can be used for browsing, video podcasting, instant messaging, emails, podcasting, video playback, word processing/office suites, sound recording, publishing, as well as others. We will also look at ways that you can use these tools in your classroom and share some examples of how teachers are currently using these in classrooms.

http://knowschools.ca/moodle/mod/book/view.php?id=1228&chapterid=389

May 21-22

PGCon

Ottawa, ON

PGCon is an annual conference for users and developers of PostgreSQL, a leading relational database, which just happens to be open source. PGCon is the place to meet, discuss, build relationships, learn valuable insights, and generally chat about the work you are doing with PostgreSQL. If you want to learn why so many people are moving to PostgreSQL, PGCon will be the place to find out why. Whether you are a casual user or you've been working with PostgreSQL for years, PGCon will have something for you.

http://www.pgcon.org/2008/

UPCOMING EVENTS

May 25-27

SMARTlinkages 2009

Kelowna, BC

SMARTlinkages brings together hundreds of industry leaders, government leaders, research leaders and students to showcase Canada's ingenuity, innovation and leadership in information and communications technology. The annual conference is a dynamic venue where people interact, companies meet brilliant new employees, where our best and brightest students showcase their talents and ideas, where government executives take the pulse of innovation, and where deals are done.

http://www.aigicrvis.ca/program2009/SMARTLinkages2009/index.html

May 29

DemTech

Montreal, QC

DemTech 2009 will showcase cutting edge projects that use information technology to encourage citizen access and foster democratic participation. DemTech is a pre-conference of the 2009 Annual Conference and Trade Show of the Canadian Library Association, sponsored by Apathy is Boring, VisibleGovernment.ca and members of the CivicAccess.ca community.

http://demtech.ca/

NEWSBYTES

March 18

The Law Society of British Columbia Goes Live with Evergreen

Vancouver, BC

The Law Society of British Columbia has gone live with the Evergreen open source library automation software. Jeremy Buhler, a graduate student from the School of Library, Archival and Information Studies at the University of British Columbia, did most of the work developing the main page and migrating data. The Law Society of British Columbia is the governing body of the legal profession in BC.

http://www.esilibrary.com/esi/news item.php?id=91

March 23

Eclipse Announces First Release of Swordfish

Ottawa, ON

The Eclipse Foundation announced today the first release of Swordfish, a next-generation enterprise service bus that provides the flexibility and extensibility required by enterprises to successfully deploy a service-oriented architecture strategy. Swordfish is based on the OSGi standard and builds upon successful open source projects, including Eclipse Equinox and Apache ServiceMix.

http://www.eclipse.org/org/press-release/20090323_swordfish.php

General Chair
Michael Weiss
Carleton University
weiss@sce.carleton.ca

PC Co-Chairs
Gilbert Babin
HEC Montréal
Peter Kropf
Université de Neuchâtel

Program Committee
Kamel Adi, Canada
Esma Aïmeur, Canada
Daniel Amyot, Canada
Gilbert Babin, Canada
Tony Bailetti, Canada
Sarita Bassil, USA
Morad Benyoucef, Canada
Vincenzo D'Andrea, Italy
Peter Emmel, Germany
Michael Franz, USA
Jaap Gordijn, The Netherlands
Ruediger Grimm, Germany
Martin Hepp, Germany
Paul Hofmann, USA
Dietmar Jannach, Germany
Gregory Kersten, Canada
Ferhat Khendek, Canada
Peter Kropf, Switzerland
Craig Kuziemsky, Canada
Anne-Françoise Le Meur, France
Luigi Logrippo, Canada
Simone Ludwig, Canada
Hafedh Mili, Canada
Morteza Niktash, Canada
Liam Peyton, Canada
Roy Rada, USA
Christoph Rensing, Germany
Alain Sandoz, Switzerland
Carlo Simon, Germany
Michael Spahn, Germany
Jun Suzuki, USA
Thomas Tran, Canada
Guy Tremblay, Canada
Petko Valtchev, Canada
Marie-Hélène Verrons, France
Michael Weiss, Canada
Yuhong Yan, Canada
Christian Zirpins, UK

Organization

Carleton UNIVERSITY
uOttawa
UQÀM Université du Québec à Montréal
HEC MONTRÉAL
Université de Neuchâtel unine

Contact
All queries to the Organizing Committee should be sent to:
weiss@sce.carleton.ca

For More Information
http://mcetech.org/

CALL FOR PARTICIPATION

MCeTech
4th Int. MCETECH Conference on eTechnologies
4-6 May 2009, Ottawa, Canada

The Internet pervades many of the activities of modern societies and has become the preferred medium for the delivery of information and services. The successful implementation of Internet applications, ranging from eBusiness, to eEducation or to eGovernment, is a multi-faceted problem, involving technological, managerial, economic, and legal issues.

The **4th International MCETECH Conference on e-Technologies** aims to bring together researchers and practitioners interested in exploring the many facets of Internet applications and technologies. Contributions and presentations focus on original and inter-disciplinary approaches to these problems and combine technological aspects with economic, managerial, and organizational aspects.

This year's conference theme is **Innovation in an Open World**, and the program is composed of a keynote presentation, scientific paper presentations, a discussion panel, tutorials, workshops, and other related events. The conference also includes an industrial track, providing a forum for practitioners to present problems and case studies that have benefited from, or could benefit from, Internet technologies in their business.

E-Health Workshop and TIM Lecture Series (May 4)
Two associated events will run on the first days of the conference:
- E-health Workshop: Towards System Interoperability through Process Integration and Performance Management
- TIM Lecture Series of the Technology Innovation Management program at Carleton University

Tutorials (May 4-6)
For practitioners, eight very interesting tutorials presented by experts will run concurrently with the workshops and paper presentations:
- Understanding how RFID Technologies & EPC Network Enable Innovative e-Business Models: a BPR Approach *(Ygal Bendavid, École Polytechnique de Montréal)*
- Social Web, Web Architecture and the OpenSocial Standard *(Claude Coulombe, U. de Montréal)*
- Interoperability in Healthcare *(Norm Archer, McMaster University)*
- Open Source Software Licensing Best Practices in a Post Jacobsen v. Katzer World *(Thomas Prowse, Gowlings)*
- Putting Zotero to Work: Free and Open Source Research Management for You and Your Institution *(Trevor Owens, George Mason University)*
- User Requirements Notation for Business Processes *(Alireza Pourshahid, IBM and U. Ottawa)*
- Technology Innovation *(Andrew Fisher, Wesley Clover)*
- Securing Internet Applications - Why SSL is Not Enough *(Preeti Raman, Carleton U.)*

Scientific and Industrial Paper Sessions (May 5-6)
A total of 27 papers will be presented in eight sessions on:
- Inter-Organizational Processes (I and II)
- Service-Oriented Architecture
- Open Source and Open Environments
- Security and Trust
- Internet-Based Collaborative Work / eEducation
- Industrial Experience
- Short Research Contributions

Registration
Please visit **http://mcetech.org/** for registering to the conference, workshops and tutorials. Early rates are available until **April 3rd, 2009**, but participants can still register afterwards (and even at the conference). However, please register early to help the organizers better plan the event.

Talent First Network
http://www.talentfirstnetwork.org/

Carleton UNIVERSITY
http://www.carleton.ca/

École de gestion TELFER School of Management
http://www.telfer.uOttawa.ca/

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

May 2009:	Open Source in Government Guest Editor: James Bowen University of Ottawa
June 2009:	Women in Open Source Guest Editor: Rikki Kite LinuxPro Magazine
July 2009:	Collaboration Guest Editor: Stephen Huddart J. W. McConnell Foundation
August 2009:	Tech Entrepreneurship
September 2009:	Business Intelligence Guest Editor: Mike Andrews SQLPower

Formatting Guidelines:

All contributions are to be submitted in .txt or .rtf format.

Indicate if your submission has been previously published elsewhere.

Do not send articles shorter than 1500 words or longer than 3000 words.

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.

Include a 2-3 paragraph abstract that provides the key messages you will be presenting in the article.

Any quotations or references within the article text need attribution. The URL to an online reference is preferred; where no online reference exists, include the name of the person and the full title of the article or book containing the referenced text. If the reference is from a personal communication, ensure that you have permission to use the quote and include a comment to that effect.

Provide a 2-3 paragraph conclusion that summarizes the article's main points and leaves the reader with the most important messages.

If this is your first article, include a 75-150 word biography.

If there are any additional texts that would be of interest to readers, include their full title and location URL.

Include 5 keywords for the article's metadata to assist search engines in finding your article.

Copyright:

You retain copyright to your work and grant the Talent First Network permission to publish your submission under a Creative Commons license. The Talent First Network owns the copyright to the collection of works comprising each edition of the OSBR. All content on the OSBR and Talent First Network websites is under the Creative Commons attribution (http://creativecommons.org/licenses/by/3.0/) license which allows for commercial and non-commercial redistribution as well as modifications of the work as long as the copyright holder is attributed.

The OSBR is searching for the right sponsors. We offer a targeted readership and hard-to-get content that is relevant to companies, open source foundations and educational institutions. You can become a gold sponsor (one year support) or a theme sponsor (one issue support). You can also place 1/4, 1/2 or full page ads.

For pricing details, contact the Editor dru@osbr.ca).

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

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.