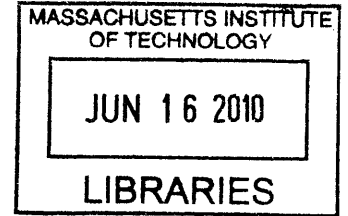


# Comparing and Contrasting Web Services and Open Source

by  
Jeremy Lee Katz



Submitted to the System Design and Management Program  
in partial fulfillment of the requirements for the degree of  
Master of Science in Engineering and Management

at the

**ARCHIVES**

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2010

© Jeremy Lee Katz, MMX. All rights reserved.

The author hereby grants to MIT permission to reproduce and  
distribute publicly paper and electronic copies of this thesis document  
in whole or in part.

Author .....  
System Design and Management Program  
February, 2010

Certified by .....  
Michael Cusumano  
Sloan Management Review Distinguished Professor of Management  
Thesis Supervisor

Accepted by .....  
Patrick Hale  
Director, System Design and Management Program



# Comparing and Contrasting Web Services and Open Source

by

Jeremy Lee Katz

Submitted to the System Design and Management Program  
on February, 2010, in partial fulfillment of the  
requirements for the degree of  
Master of Science in Engineering and Management

## Abstract

Software can either be developed in a way such that the source code is available to others, open source, or such that it is not, closed source. Open source software has a number of architectural advantages over traditionally developed closed-source software including modularity, a frequent release pattern and a strong culture of reuse.

As there has been a shift away from developing software that runs locally to a model where service based computing and web services are some of the most important software used on a daily basis by people, there has been a shift away from developing such software as open source.

This thesis looks at a comparison between open source and web services and shows how they compare on some of the aspects which are the most important architectural advantages of open source. This examination is based on a look at literature and specific web services. Through this examination, it shows that many of the benefits of open source can be found as a result of other architectural characteristics of web services.

Thesis Supervisor: Michael Cusumano

Title: Sloan Management Review Distinguished Professor of Management



# Contents

<b>1</b>	<b>Introduction</b>	<b>13</b>
1.1	Background and Motivation . . . . .	13
1.2	Approach . . . . .	14
1.3	Structure of thesis . . . . .	14
<b>2</b>	<b>An Overview of Open Source</b>	<b>17</b>
2.1	Software Binaries vs Source . . . . .	17
2.2	History of Source Code Availability . . . . .	18
2.3	The Free Software Foundation . . . . .	19
2.4	“Open Source” is Born . . . . .	19
2.4.1	Open Source Successes . . . . .	20
2.4.2	Open Source Licenses . . . . .	21
2.4.3	Open Source in a Non-Software Context . . . . .	22
2.5	Benefits of Open Source . . . . .	22
<b>3</b>	<b>An Overview of Web Services</b>	<b>23</b>
3.1	History of Computing . . . . .	23
3.2	Growth of the Internet . . . . .	24
3.3	Computing as a Service Returns . . . . .	25
3.3.1	Platforms and Web Services . . . . .	26
3.3.2	Web Applications . . . . .	29

3.3.3	A Fuzzy Line . . . . .	30
3.3.4	Benefits of Web Services . . . . .	30
<b>4</b>	<b>Modularity</b>	<b>33</b>
4.1	What is Modularity? . . . . .	33
4.2	Modularity in Open Source . . . . .	34
4.2.1	The Increasing Modularity of Mozilla . . . . .	34
4.3	Modularity in Web Services . . . . .	35
4.3.1	Modularity and Service Oriented Architecture (SOA) . . . . .	36
4.3.2	Modularity in Salesforce.com . . . . .	37
<b>5</b>	<b>The Bazaar Model</b>	<b>39</b>
5.1	What is the Bazaar Model? . . . . .	39
5.2	The Role of Bazaar Style Development in Open Source . . . . .	40
5.2.1	The Bazaar Model and the Linux Kernel . . . . .	41
5.3	The Role of Bazaar Development in Web Services . . . . .	42
5.3.1	Amazon Web Services and the Bazaar Model . . . . .	44
<b>6</b>	<b>Reuse</b>	<b>47</b>
6.1	What is Reuse? . . . . .	47
6.2	Reuse in Open Source . . . . .	48
6.2.1	Reuse in the Google Chrome Web Browser . . . . .	49
6.3	Reuse in Web Services . . . . .	50
6.3.1	Reuse of Facebook Web Services . . . . .	51
6.3.2	Facebook's Reuse of Open Source . . . . .	53
<b>7</b>	<b>Economics</b>	<b>55</b>
7.1	Making Money with Open Source . . . . .	55
7.1.1	Making Money as an Open Source Software Company . . . . .	56
7.1.2	Making Money with Open Source as a Complement . . . . .	57

7.2	Making Money with Web Services . . . . .	58
7.2.1	Making Money with Consumer Web Services . . . . .	59
7.2.2	Making Money with Business Web Services . . . . .	59
<b>8</b>	<b>Conclusion</b>	<b>61</b>
<b>A</b>	<b>Facebook Connect Sample Code</b>	<b>65</b>





# List of Figures

3-1	IBM System/360 computer from 1964. Image courtesy of Computer History Museum.[Computer History Museum] . . . . .	24
4-1	Elements of a service oriented architecture[Lindner]. . . . .	37
5-1	The release dates of various Linux kernel versions[Kernel.org] . . . . .	42
6-1	MotionBased and Facebook . . . . .	52
6-2	Before logging in, the user sees a <i>Connect</i> button . . . . .	53
6-3	The user allows the site to use their Facebook identity . . . . .	53
6-4	The site can then show the user's Facebook identity and use it . . . . .	54
A-1	CNN and Facebook Connect . . . . .	66



# List of Snippets

2-1	Sample Source Code . . . . .	18
3-1	An HTTP GET Request . . . . .	25
3-2	An HTTP GET Response] . . . . .	25
3-3	A Simple SOAP Request . . . . .	27
3-4	A Simple SOAP Response . . . . .	28
3-5	A Simple REST Request . . . . .	28
3-6	A Simple REST Response . . . . .	29
A-1	Facebook Connect sample source code . . . . .	67
A-2	Facebook connect security snippet . . . . .	68



# Chapter 1

## Introduction

### 1.1 Background and Motivation

I have been involved both professionally and personally in the world of open source development for a long time. This involvement has led to a great passion for the ideals of the open source community that are based around the availability of software source code for purposes of education as well as to modify and extend for my own needs.

There has been a shift over the course of the decade away from the distribution of software to be run on local computers to a world in which much of the most critical software applications used by people are instead accessed on remote servers. With this shift has also come a standard for not having the source code of such applications available. This concerns me for future software developers who may not have access to a rich set of open source applications to inspect, modify and improve.

Also, I have started at a new job during my time at MIT where I will be much more directly involved in the use of web services in the development of a larger web based application.

Therefore, I hope that a comparison of some of the architectural characteristics of open source and web services will help me to make that professional transition as well as showing that some of the properties of open source can be found in web services

even without the source code being made available.

## **1.2 Approach**

My approach to the research necessary for this thesis was largely based upon my own experiences and an extensive literature review.

My own experiences are drawn from over a decade of work as an open source developer contributing to a wide variety of projects as well as leading the development on several of open source projects. This gave me a good foundation to start from on the intricacies of how open source works and an instinct for what some of the most important architectural characteristics were.

For a literature review, I consulted a variety of sources that are publicly available. These included a number of articles and theses as well as a wide variety of Internet publications. Many of these Internet publications have also had alternate methods of distribution including presentations at conferences or printed versions of the content.

I also have tried to let these findings influence my full-time employment and tried to learn from both the results of applying the knowledge as well as applying knowledge gained from the job to the contents of this thesis.

## **1.3 Structure of thesis**

To organize this thesis, I first take a look at an overview of and the history which has led to the growth of both open source and web services. This history and overview can be found in Chapters 2 and 3.

From there, there are a set of chapters in the middle looking at three important characteristics which can be found in both open source and web services: modularity, the bazaar model of development, and reuse. As a final point of comparison, there is a look at some of the challenges faced by both open source and web services in

Chapter 7.

Finally, Chapter 8 pulls all of these together wrapping up to show how web services share architectural properities with open source without the need for the release of source code.





# Chapter 2

## An Overview of Open Source

### 2.1 Software Binaries vs Source

Open source software is, at the most basic level, computer software for which the source code is available to people other than the original developer.

To execute software on your computer, you require the software in a format which can be interpreted by the actual hardware; this is generally referred to as *machine code*. In general, this is how most software is distributed. While convenient and understandable by the computer, it is not a format which is really consumable by humans as it is a stream of very simple, low-level instructions. Writing software using just machine code would be an extremely time-consuming and error-prone process.

Instead, most computer programmers write software using a higher level language with statements that can be more easily read and understood (see Snippet 2-1). This higher level language code is then compiled, or translated, into the lower-level machine code. This code is referred to as the *source code* and can be written in any of a number of languages including C, C++ and more. In some cases, the source code is in an even higher level language such as Java or Python which then gets processed by an intermediate interpreter to turn the source code instructions into machine code at runtime.

```
int main() {
    printf("Hello World!");
}
```

**Listing 2-1:** Simple Hello World source code written in the C programming language

For open source software, in addition to the distribution of the compiled machine code as with most software, the source code code is also made available and able to be studied or modified.

## 2.2 History of Source Code Availability

As the computer age dawned in the 1950s, this availability of source code was seen as the normal case. Software was, at the time, not seen as an artifact with any inherent monetary or intellectual property value. It was instead seen by vendors such as IBM as a tool to take advantage of the computing hardware which had a clearly high monetary value. The availability of software source code allowed users to extend and modify the vendor-provided functionality as they needed.

There began to be a shift away from such openness as computing platforms were introduced during the 1960s which were targeted to more users. The source code originally still had to be available due to the customization needs, but there were frequently contractual restrictions restricting disclosure or sharing.

With the introduction of the personal computer in the 1970s, the final step away from source code availability was made as software made the final transition from a tool to a good with a significant monetary value and very large businesses have been formed around the development and sale of software outside of the benefits by selling other things[Campbell-Kelly, 2008].

## 2.3 The Free Software Foundation

This shift was not without its opponents, though. Some who had been involved in computing since its earlier days were upset with this movement away from sharing seeing it as detrimental to the furthering of computer science as a field. Most famously, Richard Stallman founded the Free Software Foundation in 1985 in an effort to encourage cooperation and the sharing of software, including its source code[Stallman, 2008].

Free software refers to software which is *free* as in *libre* and not necessarily free in terms of cost. This freedom is embodied in the four freedoms which the Free Software Foundation calls for

1. To run the program for any purpose
2. To study how the program works and modify it. This requires access to the source code of the program
3. To redistribute copies
4. To improve and distribute your improved version

While the Free Software Foundation and the associated GNU Project, an attempt to build an entirely free operating system, made progress on their goals to develop an entirely free system, the primary interest in it came from those involved in computer science research and not business. Some usage of free software in a business context existed with firms such as Cygnus Solutions, Red Hat and others but it was the exception rather than the rule.

## 2.4 “Open Source” is Born

One problem with the adoption of the ideals of free software was the use of the term “free”. Businesses were, not unreasonably, concerned about the confusion involved in

trying to sell software which was also marketed as “free”. The difficulty is two-fold, both in that people do not expect to pay for something which is “free” and also there is a negative stigma in terms of quality associated with things which are “free”.

This concern came to a head in early 1998 as Netscape was beginning to look at the idea of releasing the source code to their web browser software, Netscape Navigator, in an attempt to help with their struggle against the growth of Microsoft’s Internet Explorer browser. A group of the thought leaders in the free software world, notably without Richard Stallman, met and came up with the term “open source” as a term to communicate many of the same ideas but without the negative, for business, connotations of the word “free”[, OSI].

While Free Software was focused on the idea that source code should be available only as a step towards satisfying the four basic freedoms, open source was instead focused most strongly on the access to the source code used for the program as well as the ability to improve and build upon such software. While the official definition of open source from the Open Source Initiative[, OSI] includes the same aspects of freedom as the four freedoms called for by the Free Software Foundation, they are not the primary focus of the document. This change of focus away from freedom has helped greatly in making the idea of open source more attractive for businesses to use.

### **2.4.1 Open Source Successes**

The list of successful open source projects could itself fill a substantial volume, but the past decade has seen a massive growth in adoption of an open source philosophy while developing software. A short list includes

- The Linux operating system kernel is used in everything from embedded devices all the way up to supercomputers as the foundation for the system.
- The Apache httpd web server powers nearly half of the web sites running on

the Internet today[Netcraft Ltd, 2009].

- The MySQL database server is used for many database-backed web sites and was deemed important enough to be purchased by Sun Microsystems in early 2008.
- The Eclipse development environment was originally developed by IBM but has since been released as open source and is a fully-featured and community developed development environment for Java and other programming languages.
- The Firefox web browser is the evolution of the original Netscape Navigator source code released by Netscape in 1998 and is currently used for over 45% of web browsing[Refsnes Data, 2009].

## 2.4.2 Open Source Licenses

Even under the umbrella of “open source” there is a great variety in exactly what is meant and required. This variety is largely reflected by the various licenses which exist and are used to govern the distribution and modification of open source software. At one end of this are the licenses promoted by the Free Software Foundation including the GNU Public License (GPL) which require that you include the source code if you distribute the software with any modifications. Other licenses such as the BSD and MIT licenses, which originated at the University of California at Berkeley and MIT respectively, have no requirements to distribute your changes in source code form. While this variety of terms is slightly confusing, it has also helped to ensure that open source software can be used in a wide variety of circumstances where there may be other constraints that have to be considered.

### 2.4.3 Open Source in a Non-Software Context

Given the success seen with open source software, there has been a movement to try to apply the term “open source” to other, non-software based contexts. These include things such as MIT’s Open Courseware system providing access to open educational materials or the community developed and edited encyclopedia Wikipedia. For the purposes of this thesis, though, I will focus purely on the advantages of open source in the context of software.

## 2.5 Benefits of Open Source

As the adoption of open source software has grown, there have been efforts to try to determine if there are characteristics which make this growth likely. From this work, there have been a number of things which have an overall impact on the architecture of the software system which seem to be emergent as a result of developing a piece of software as open source. A few of the most notable of these include:

- Modularity
- Bazaar development
- Reuse, both in terms of reusing specific code elements as well as by providing a platform to build upon
- Symbiotic relationship for companies [Feller et al., 2005]

In the later chapters of this thesis, each of these will be looked at in greater detail.

# Chapter 3

## An Overview of Web Services

### 3.1 History of Computing

The rise of computing as a service is not really a new idea or paradigm. In the early days of computing, all computing was basically “computing as a service” where there were large server computers located somewhere. Users then interacted with these computers “remotely”, at first via physical interfaces such as punch cards and later via terminals or other electronic interfaces. The concept of running software “locally” was somewhat unheard of as computers were very large machines taking up entire rooms, or even floors in some cases!

It was only in the late 1970s and early 1980s that the idea of a personal computer and running software locally began its rise to prominence as a result of the invention of the microprocessor. This movement to personal computers was a substantial shift in the dominant paradigm of computing at that time. Many of the companies that we see today as large or important computer companies such as Microsoft, Apple, and others got their start during this era by seeing the growth opportunity and jumping on it. With this and later advancements such as graphical user interfaces like the Macintosh and Windows, there was a great growth in the importance of software running on a machine, which was “local”.



Figure 3-1: IBM System/360 computer from 1964. Image courtesy of Computer History Museum.[Computer History Museum]

In doing so, users were able to have more intuitive interactions with the software they were running the machine via things like a mouse. And as the computing power of microprocessors continued to increase, this became even more viable for computing. The later growth of laptops began to allow for such interactions to occur while not tethered to a desk and be carried with you wherever you went. The dominant usage mode for the computer had changed.

## 3.2 Growth of the Internet

In the 1990s, a new mode of computing began to rise in importance as the Internet grew from infancy to an incredibly important resource. While the Internet's roots can be traced to work on ARPAnet in the 1980s, ARPAnet was limited in its usefulness to a small class of computing users at the time.

The invention of the World Wide Web and the Hypertext Transfer Protocol (HTTP) by Sir Tim Berners Lee in the early 1990s began the growth of a valuable network resource for end users. Originally conceived as a way for people to communicate and share information, in many ways it is responsible for the expansion



and growth of the Internet.

HTTP is a relatively simple, text-based protocol involving requests from a client to a server and then a response from that server. One of these requests is the **GET** request which asks the server to send a given file as its response. Another type of request is the **POST** request which also provides some data which is meant to be processed by the server before getting back a response. Simple examples of a **GET** request and the corresponding response can be found in Listings 3-1 and 3-2.

```
GET / HTTP / 1.1
Host: www.myhost.com
```

**Listing 3-1:** An HTTP GET Request

```
HTTP/1.1 200 OK
Date: Thu, 19 Nov 2009 22:38:34 GMT
Server: Apache/2.2.13 (Fedora)
Connection: close
Content-Type: text/html; charset=UTF-8

hello world!
```

**Listing 3-2:** An HTTP GET Response]

In the early days of the Web, the primary usage was for sharing and displaying static information which was retrieved from websites rather than any form of interactive applications. Much of the effort through the decade was spent on concerns of interoperability, ease of information transfer and security[Berners-Lee, 1996]. These efforts were important, but kept the Web as a largely read-only environment.

### 3.3 Computing as a Service Returns

To a small degree at the very end of the 1990s but especially over the course of the first decade of the twenty-first century, there has been a movement towards a more interactive and immersive Web experience. Tim O'Reilly is well-known for having

made a statement that a change was underway from the original *Web 1.0* world of non-interactive websites to a new world of interactive web sites which he referred to as *Web 2.0*[O'Reilly, 2005].

This change has been characterized by a shift towards more things hosted on the Internet which can be perceived as platforms or applications as opposed to pages listing information or simple e-commerce shops. This is interesting as it allows people's interactions with their computers to change back to where the local computer is just more of a terminal interacting with a remote server as opposed to having all of your important applications running locally on your own machine.

This has also been helped by the growth of the mobile Internet. The drive to be able access one's data from anywhere from your computer to your cell phone has led to a need for easily accessible services. The ubiquity of the HTTP protocol and the ease of implementing access with it on any platform has made it an obvious choice for this transition of computing services.

### **3.3.1 Platforms and Web Services**

The primary part of computing as a service today which will be focused on for this thesis are the platforms or web services which are then used by others to build their own applications. As the patterns for building the platforms have matured, there has been a convergence around basing these platforms and services around the protocol which Tim Berners Lee initially developed and built the World Wide Web on, the HyperText Transfer Protocol.

The World Wide Web Consortium (W3C) defines a web service as

A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML

serialization in conjunction with other Web-related standards.” [Booth et al., 2004]

As stated in the definition, SOAP is used over HTTP and takes advantage of the previously described **POST** request type. The data in this case is an XML-encoded set of data that can then be processed by the server. By way of example, one could have a web service implemented using SOAP to get the price for a given item in an online shopping website as seen in Snippet 3-3.

```
POST /Price HTTP/1.1
Host: www.mystore.com
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Body xmlns:m="http://www.mystore.com/price">
  <m:GetItemPrice>
    <m:ItemName>Can of Soup</m:ItemName>
  </m:GetItemPrice>
</soap:Body>
</soap:Envelope>
```

**Listing 3-3:** A Simple SOAP request for the price of a can of soup

Upon receiving this request, the server would parse the XML-based SOAP request and see that there was a desire to run the *GetItemPrice* method for an *ItemName* of *Can of Soup*. This method could be implemented in any way on the server side including a database lookup or anything else. The server would then send back a response which is also SOAP encoded as seen in Snippet 3-4.

As it turns out, though, this definition that a web service must use SOAP is somewhat restrictive and in the five years since it was written, there have been substantial developments of other ways to build web services. These include representational

```
HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Body xmlns:m="http://www.mystore.com/price">
  <m:GetItemPriceResponse>
    <m:ItemPrice>0.99</m:ItemPrice>
  </m:GetItemPriceResponse>
</soap:Body>

</soap:Envelope>
```

**Listing 3-4:** The SOAP response for the price of a can of soup

state transfer (REST) as first described by Roy Fielding in his work on network-based software architecture [Fielding, 2000] and simpler transfer formats such as JavaScript Object Notation (json). An example of the request to and response from a REST service to find the price of a can of soup can be found in Snippets 3-5 and 3-6.

```
GET /rest/Price/CanOfSoup HTTP/1.1
Host: www.mystore.com
Accept: application/json
```

**Listing 3-5:** A Simple REST request for the price of a can of soup

As you can see, this is significantly less verbose of a protocol and as a result is significantly easier for a developer to work with. The downside is that the lack of verbosity makes it such that the protocol describes itself less and therefore requires a clearer understanding of what the interface between the components are.

Given the wide usage of both types of service, I will not restrict my usage of the term “web service” to only those services using SOAP and will instead rely on a more general definition of a web service in referring to any software system operating over

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
Content-Length: nnn

{ "price": 0.99 }
```

**Listing 3-6:** The REST response for the price of a can of soup using JSON

a network where the communication is conveyed via the HTTP protocol.

These web services can then either used by other web services to perform data operations rather than options such as database queries due to the sheer volume of data which is being worked with. Many of these web services began as internal services within companies, but they have increasingly become more public. Examples include many of the web services provided by Amazon around their online retailing business or services provided by financial trading firms to access up-to-date stock information.

### 3.3.2 Web Applications

The second part of computing as a service is the actual applications which live on the network and are built on top of the platforms and web services above. The financial tracking services provided by Mint.com, purchased by Intuit in the fall of 2009, take advantage of other companies' web services to aid in the accessing of their users' financial data. Another example that is used by many consumers are web based games inside of Facebook such as Farmville which are then able to take advantage of the social network to share your progress and achievements with your friends

Such web applications aren't limited to being used for consumer or end-user purposes, though,. There are also a number of web applications that are more targeted at usage by businesses. One such application is the HubSpot application suite for inbound marketing analysis; it makes substantial use of external web services including those offered by companies such as Google and Salesforce.com to be able to integrate

information from all of these sources and allow a marketing department to maintain and manage their funnel of incoming prospects and leads.

### **3.3.3 A Fuzzy Line**

It is worth noting that the line between a web services platform and a web application is not always clear. Applications can and frequently do provide their users with web services which can be used for additional and innovative functionality or will sometimes even provide such services to other developers.

This can be seen on end-user oriented applications such as Facebook. Many users use just the basic functionality of Facebook, but there also exists an extensive platform that other developers can build their own applications against. Today, this platform is used by some thousands of developers for many applications that extend Facebook to provide games, bring in other services and more.

Business web applications do not ignore this capability to straddle the line between platform and applications as well. One needs look no further than the trail-blazer of software as a service companies, Salesforce.com, to see a business web application which is extensively used in its own form but as previously mentioned, they also provide a variety of web services so that their users can integrate Salesforce.com with other web applications that they would like to use.

### **3.3.4 Benefits of Web Services**

The largest benefit of the growth of web services is clearly the ability to access them from anywhere. This decentralized nature is increasingly important in today's world where businesses have offices spread across the globe and need to allow access to various services from any of these offices without requiring complex computer system setups in each.

Going beyond that benefit, you can begin to see many of the same benefits as exist when developing in an open source environment. The later chapters of this thesis will

look specifically at modularity, the bazaar development model, and reuse in a web services context.





# Chapter 4

## Modularity

### 4.1 What is Modularity?

One of the major aspects of the architecture of a product or system is whether it is integral (closed) or modular (open). As described by Ulrich and Eppinger, a modular architecture is one in which each chunk or part implements a functional element entirely and the interactions between the parts are well-defined and important to the primary function of the system[Ulrich and Eppinger, 2008].

More modular systems have been shown to allow for a quicker product development cycle through the creation of platforms[Muffato and Roveda, 2000]. There are also strong gains to be had in terms of an ability to adapt to product change for any of many reasons far more easily. This is because the architecture is such that a minimum of physical changes are required to implement the functional change.

While modularity plays a strong part in the development[Baldwin and Clark, 1997] of many more complex systems, it has not been shown to be a necessary precondition for such systems. Especially in arenas where the driving factors are not related to the flexibility of the product, you are more likely to see integral products. Also, integral systems can tend to have a higher performance than their modular counterparts due to being able to avoid expensive bottlenecks at module boundaries.

Another way to look at it is that integral systems are made up of very strongly coupled subsystems where modular systems, instead, are much more decoupled or decentralized. This makes it such that switching out parts of an integral system ranges from difficult to impossible. On the other hand, well-defined interfaces make it relatively straight-forward to do so in a more modular system.

It is important to note that modular vs integral systems is not a black or white comparison; most systems have elements of both present. But at the same time, it is possible to compare the modularity of systems and have a reasonable area about which to discuss.

## **4.2 Modularity in Open Source**

This decoupled or modular nature is an architectural characteristic which is strongly present within open source. While some open source software projects make design choices which explicitly lead to such modularity, even among those which do not, there is a strong bias towards such modularity.

### **4.2.1 The Increasing Modularity of Mozilla**

Alan MacCormack, et al performed a study to look at the modularity of open source and especially how it compares to that of the more traditional closed source software[MacCormack et al., 2006]. They used a Design Structure Matrix (DSM) to look at dependencies of projects and used that DSM to assess the complexity and modularity of the system. While this does not allow the assignment of any sort of real “number” representing the modularity, it does allow for the comparison between systems by looking at the various complexity characteristics of the DSMs developed.

In the study, they compared the Linux kernel and the initial release of the Mozilla web browser. As previously mentioned, the Mozilla web browser was released by Netscape in 1998 as the open source version of their Netscape Navigator browser.

Given the state of the source code at that time, MacCormack felt that it was a reasonable proxy for a closed source developed software system, especially given the difficulty in attaining the source code to closed source software. This initial comparison showed a significantly more integral design being present in the Mozilla browser as compared to the Linux kernel.

This left some questions, though, as to whether there were inherent differences in the modularity of an operating system and that of a web browser. Therefore, the researchers went a step further and looked at the source code of the Mozilla web browser after its first significant redesign as an open source project. Comparing this post-redesign and now open source architecture showed a significantly more modular design than had been seen with the original source code release of Mozilla. Also, it was a comparable modularity to that of the Linux kernel.

This led them to conclude that the software domain did not actually make a significant difference in the modularity of the system. Instead, a modular and loosely coupled system is a necessary outcome for the collaborative and distributed set of developers which work on open source projects. In contrast, software systems which are developed in a more proprietary fashion can be more strongly coupled. Some of the problems of this more integrated architecture are offset by an ability to solve problems through in-person interactions and sharing of information.

### **4.3 Modularity in Web Services**

Given the inability to assign a concrete number defining the modularity of a system, it is difficult to directly compare the modularity of web services to that of open source. But one can look at the modularity of a web service based system vs a more traditional system. While no formal study has been done, in early 2003 when Salesforce.com was a relative newcomer to the Customer Relationship Management (CRM) market compared to long-time industry heavyweights Oracle, Siebel and SAP, it was reported

on how much more modular Salesforce.com was than the incumbents[Sweeney, 2003]. Since that time each of Oracle, Siebel and SAP have worked towards turning their CRM solutions into web services and thus have increased in modularity.

Looking at a more abstract level, though, the ecosystem surrounding web services tends to strongly emphasize modularity. This begins even at the definition of a web service which includes the requirement of a definition of the service being provided. This interface definition in turn sets an expectation for other components on how they will interact with the system. These expectations in turn help to keep clear boundaries between different parts of the system, which is one of the key aspects of modularity.

### **4.3.1 Modularity and Service Oriented Architecture (SOA)**

One of the most prominent and highly used patterns in web service development is Service Oriented Architectures (SOA). Generally used in internal business applications to help tie various backend systems together, SOA involves having a number of web services with their own functions to perform being pulled together to create an even larger system.

The Organization for the Advancement of Structured Information Standards (OASIS) is a group which helps to drive the adoption and definition of various web service standards, especially within the domains of business. They define a Service Oriented Architecture as

A paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains. It provides a uniform means to offer, discover, interact with and use capabilities to produce desired effects consistent with measurable preconditions and expectations.[OASIS, 2006]

This is a relatively complex definition but what is most important to recognize

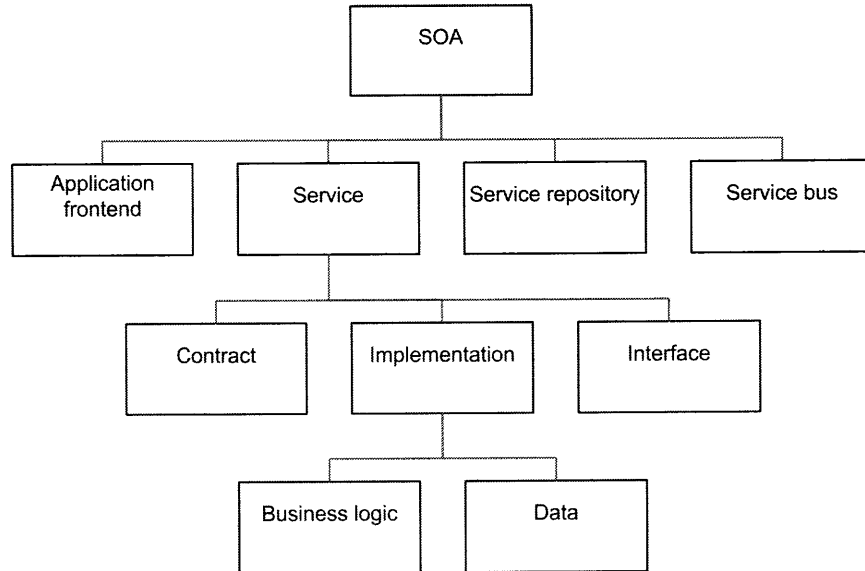


Figure 4-1: Elements of a service oriented architecture[Lindner].

from it is the fact that SOA relies on having distributed capabilities as well as discoverability around those capabilities. These are in fact the very characteristics that were described earlier as required for a modular system. And in fact going beyond the definition, best practices for building such an architecture state that the interfaces between modules should be explicitly published in a standard format[Endrei et al., 2004].

### 4.3.2 Modularity in Salesforce.com

Switching from the generalized case of modularity as promoted by Service Oriented Architectures, we can look at the modularity present in Salesforce.com

Salesforce.com is probably one of the most well-known, if not best known, software as a service providers. The company was founded in 1999 and provides a variety of customer relationship management (CRM) functionality all as hosted software available via the web rather than software which is packaged and provided to customers. Today, they have over 60,000 customers and are a publicly traded company with a nearly eight billion dollar market cap[Salesforce.com, 2009a].

Salesforce.com has been a pioneer in the world of software as a service and have had a strong bias towards allowing for large amounts of customization within their product to help meet the widely varying needs of their customers. While some concepts are applicable to all customers looking for a CRM tool such as customers, leads, and salespeople, exactly how those objects interact and are connected is not common. To help meet these needs, they have built a system made up of a large number of modules that can be used together as needed for your specific business. These modules “can be assembled with minimal coding in building-block fashion.” [Salesforce.com, 2009b]. This ability to use modules as building blocks is enabled by having a well-defined interface between them.

In addition to the customization of their own application, Salesforce.com provides a rich set of web services interfaces that can be used not only to work with their product but also to effectively tie it together with other products. These web services are generally provided via both SOAP and REST interfaces as described in Chapter 3 and are documented with a standard interface file. These interfaces can either be specific to the business of a customer to tie in with other backend processes that they already have as part of a larger Service Oriented Architecture or they can also be more general such that they can be used by other software.

This more general method of interacting with Salesforce.com is used from within the HubSpot inbound marketing platform. With the functionality provided, HubSpot is able to create leads within Salesforce.com that can be assigned to salespeople for a follow-up and then also keep a general view of the state of those leads and prospects within the view provided for tracking the effectiveness of your inbound marketing. This integration is valuable to HubSpot’s customers as it allows them to have a complete view of their sales and marketing efforts which is trackable so that they can determine which marketing efforts are the most effective.

# Chapter 5

## The Bazaar Model

### 5.1 What is the Bazaar Model?

The concept of a bazaar development model is an idea originally presented by Eric S. Raymond at the Linux Kongress conference in 1997 as a further evolution of open source. The idea has since been refined and further published in his essay *The Cathedral and the Bazaar* [Raymond, 1999a]. In the essay, he contrasts a bazaar model in which developers develop their software in a very public fashion which encourages participation in the process of software development with that of the, more traditional, cathedral style in which development happens in the background and there are then periodic, and typically infrequent, releases that users can consume. Even within the world of open source, when the paper was written, many projects were developed more in line with the cathedral style and just had source code accompanying the infrequent software releases.

As an outcome of the bazaar model, there are two substantial propositions which have held up over the past decade with the growth of open source as the main advantages of the bazaar model.

The first of these is “given enough eyeballs, all bugs are shallow” [Raymond, 1999b]. Raymond terms this *Linus’s Law* after Linus Torvalds, the creator of the Linux op-

erating system. The idea behind this proposition as originally stated is that with more eyes looking at your system, bugs will be found more quickly. An important corollary of the idea is that fixing of problems is less difficult of a problem than the original act of finding the problems. Raymond cites this as the fundamental difference between cathedral and bazaar style development. He asserts that due to a lack of this property, cathedral style development sees problems and bugs as difficult things which lead to the long release cycles frequently seen in cathedral style development.

The second very complementary proposition provided by Raymond is that of “release early, release often”. The idea behind this is relatively straight-forward. Essentially you wish to release your software early in its life and then continue to release it frequently as it evolves. By doing so, you make it easier to get feedback earlier in the life-cycle of the project. As has been shown in the standard cost of change curve, this in turns reduces the cost of making any needed changes and can also be shown to improve the quality of software. This is something of a pre-condition for the above idea of more eyeballs making bugs shallow as you cannot get the eyeballs on what you are working on without at least some type of release.

## **5.2 The Role of Bazaar Style Development in Open Source**

Clearly, given the original motivation for Raymond’s work, open source is a prime example of where the bazaar model of development occurs. It has become something of a core tenet of a large number of open source projects including of course the Linux kernel, many of the open source distributions including Ubuntu, Fedora and OpenSuSE and the X Window System. From these examples, though, one might think that the practice is limited to relatively large and in some ways mature projects. This isn’t the case, though. Just a casual look through some of the projects present on the popular open source hosting site SourceForge shows examples such as the Alleyoop



memory profiler front-end[AlleyOOP] where some of the release news includes the idea of living by a philosophy of releasing early and often or the Bluefish editor[Bluefish] where they claim having received many benefits as a result of adhering as a result of releasing early and often.

### **5.2.1 The Bazaar Model and the Linux Kernel**

Perhaps the single best example when talking about bazaar style development is of course the Linux kernel. Since the earliest days of its development, the Linux kernel has been developed in the bazaar style. The earliest release of Linux by Linus Torvalds in 1991 was stated to be a very immature release but Linus believed that the value in getting the initial release out there and viewed by more people was more important than it being “done”. After this first 0.01 release, Linus made additional releases on a relatively frequent basis.

As Linux began to be developed by more people, there came to be a split into having two actively developed kernels at any one time. The first was the “stable” kernel series into which smaller changes went. The second was a “development” kernel series into which larger, more experimental, and more destabilizing changes went. The development kernel was released on a very fast basis, sometimes with several releases over the course of a week and with various changes at different stages of completion.

This was effective for a while but as the cycle of releasing new versions of the stable kernel series stretched out (see Figure 5-1 simultaneously with the increasing dependence on Linux by more companies, there was a need for some kind of change. Torvalds then took things a step further than could have been imagined in 1997 in terms of releasing early and often by announcing a change in the development style in the summer of 2004[Corbet, 2004]. This change was a switch to have all development occurring in the primary, stable series of the kernel. This helped to focus all of the eyes on one code tree for development and thus, in theory, making all bugs shallower and more discoverable. Releases were still frequent and as people adjusted to the change,

Kernel Version	Development vs Stable	Release Date
v0.01	development	30 October 1993
v1.0	stable	13 March 1994
v1.2	stable	7 March 1995
v2.0	stable	9 June 1996
v2.2	stable	26 January 1999
v2.4	stable	4 January 2001
v2.5.2	development	15 January 2002
v2.5.3	development	30 January 2002
v2.5.4	development	11 February 2002
v2.5.5	development	20 February 2002
v2.6.0	stable	18 December 2003
v2.6.29	stable	23 March 2009
v2.6.30	stable	10 June 2009

Figure 5-1: The release dates of various Linux kernel versions[Kernel.org]

they actually began to come on a very regular schedule of every three months.

Even this was not quite enough for Torvalds, though. As the development of Linux grew and became more complex, there started to be more contributors who were actively working at any time on an area of the code. To help increase the visibility of what was occurring and ease the effort of everyone working together, Torvalds built the *git* distributed version control system[git] to provide an easier way to allow anyone to look at the source code being developed by any individual. While publishing unfinished changes in this manner is not the same as releasing them, it has certainly served to get even more development done publicly and thus in the spirit of the bazaar model.

### 5.3 The Role of Bazaar Development in Web Services

At first glance, web services do not seem to have any of the advantages of bazaar development as characterized by Raymond and instead are purely the result of cathe-

dral style development efforts. While there are isolated instances of open source web services that can then get the benefits of a bazaar style of development just as any other open source software system, that is the exception and not the norm for web services. Interestingly, though, as you look closer, the two primary advantages of bazaar development manifest themselves more generally with web services development even without them being developed in an open source fashion.

First of all, many web services actually end up following the ideas behind release early, release often. One of the problems with following this philosophy in more traditional and proprietary software is that it is then very difficult to keep customers running on the “current” version if they must frequently download and install a new revision of the software. With open source, this tends to be less of a concern as there is less of a support component as the users are less likely to be paying for the frequent releases. Web services are able to sidestep this problem, though, as the software is running on servers under the control of the company or organization doing the development of the web services[Rhoads Lindholm, 2007]. This makes things significantly easier to update and leads to an ability to release far more frequently.

One extreme example of this is Yahoo’s Flickr web photo sharing service. Two Flickr employees gave a presentation on their development and deployment practices at the O’Reilly Velocity conference in June of 2009 where they revealed an average of over *ten* releases of the software stack per day[Allspaw and Hammond, 2009].

While not all web services are released on such an aggressive schedule, most seem to take advantage of the idea of releasing early and often. Salesforce.com puts out releases with a frequency mirroring the change of seasons[Garner, 2003]. Google also regularly rolls out changes to their many web services, generally on a schedule of when things are ready to be used.

The idea of many eyes as expressed in Linus’s Law, though, is less clearly a part of the web service development process. Certainly there are web services with large numbers of users to help in identifying issues. Therefore, the corollary regarding

the fact that finding the problems is more difficult than fixing them is covered in these services. And also, the open interfaces also at times helps in determining where problems occur. Overall, though, this is an area where web services do not receive the advantages of open source development.

### **5.3.1 Amazon Web Services and the Bazaar Model**

For a look at the use of a bazaar development model in web services, I will look at the suite of Amazon Web Services. Amazon Web Services is a set of services provided by Amazon largely to improve developer productivity, especially when building web applications[Amazon, 2009]. There are a number of services available now, but the value of releasing early and often to Amazon can be seen by focusing on one of the original services – the Elastic Compute Cloud (EC2).

The EC2 service provides computing infrastructure in an on-demand fashion to allow developers to scale out without the purchase and installation of additional hardware. While EC2 provides more than just simple web services, the functionality is all exposed via web services and is extensible as such. When originally opened to the public in late 2006, the functionality available was extremely limited. Users could launch and terminate instances, put instances into some simple security groupings and create your own images. Users were also limited to a maximum of twenty instances at any one time.

While these limitations were fairly severe and ruled out usage for some users, others were able to get started and provide very early feedback to Amazon on how the service could be improved. While releasing such an unfinished version of the product could be seen as detrimental, Amazon instead saw the value in releasing early and iterating to improve the product over time[O’Grady, 2009]. The feedback they received from the early releases coupled with the high frequency with which they were able to update the web services and environment have led to Amazon’s commanding leadership in the space of on-demand computing infrastructure.

Over time, the new functionality which has been added to these services include minor tweaks such as improved performance as well as more substantial changes including support for Windows in addition to just Linux, multiple sizes of machines being available, multiple locations for running instances and the integration of virtual private networking functionality for connecting your EC2 cloud to your own private infrastructure. They have also rolled out a completely new REST based interface in addition to the original SOAP interface which was available for interacting with the service.

This model of releasing early and often has since been continued as they have expanded the services provided via Amazon Web Services. Today the services available also include file storage (Simple Storage Service), a simple distributing queuing system (Amazon Simple Queue Service), a content distribution network (Amazon Cloudfront), a very simple database (Amazon SimpleDB), a way to take advantage of a new method of distributed computing pioneered by Google known as Map-Reduce (Elastic Map Reduce) and a full-fledged relational database (Amazon Relational Database Services). As each of these has been launched, they have had the bare minimum of functionality to be useful to someone but they have been quickly updated and increased in functionality. Given their online nature, this is able to be done without requiring the users to do anything other than start taking advantage of the new functionality.



# Chapter 6

## Reuse

### 6.1 What is Reuse?

The concept of reuse is one which is strongly linked to that of modularity. In its very simplest form, reuse is taking a component and using it within another context. The linkage to modularity comes in that the interfaces and well-defined nature of modular components make them easier to reuse in another context. More integral systems tend to be less appropriate for reuse as the components would require significant re-engineering to be useful in another context.

Applying the idea of reuse to software is something which has been an ongoing area of work and research over the history of computing. Much of the early research into software reuse looked at it from a very systemic view and showed it to be difficult to do in a very general fashion[Frakes and Isoda, 1994]. Even more recent studies[Desouza et al., 2006] have shown that reuse continues to lag as a general rule due to the difficulty involved in doing so even as the benefits of such reuse have become increasingly clear.

In the early 1990s, object oriented programming was seen as providing the solution to the problems of software reuse. Formally, object oriented programming involves classes or other types of “object”. These objects are able to be modified and reused

throughout your software so that you can write code as few times as possible. The growth of languages such as C++, Java and later C# are, to some extent, attributable to their primarily object oriented focus as opposed to the more procedural focus of earlier programming languages.

Much of the reuse that occurs within these languages has been strongly influenced by the book *Design Patterns*[Gamma et al., 1994] which is perhaps the most widely read book on software reuse and how to implement such reuse within your own software. The authors strongly stress that you should strive for ensuring that reuse within your software be more *black box* than *white box*; that is, that you should ensure that the software doing the reuse is isolated from the details of what is being reused such that internal implementation details can change without any need to change the callers.

While these have certainly increased the amount of reuse, there is not evidence that following such methodologies actually has a significant improvement on productivity in software development[Potok et al., 1999] or on software quality[Colagrosso, 1993].

## 6.2 Reuse in Open Source

Open source is a factor which tends to increase the amount of code reuse which is possible and which occurs[Haefliger et al., 2008]. By having the source code available, it becomes easier for other developers to examine and learn from the original software. This can lead to these new developers using what they have learned to develop better software.

But another far more practical point is that the licensing used for most open source software is written such that it not only allows, but encourages, other developers to take a piece of software and modify it. One of the most popular open sources licenses, the GNU General Public License (GPL)[GPL], not only encourages such modifications but then requires that developers who distribute such modified versions must also



distribute the modified source code for others to read and learn from. The modified work can either be distributed described as a modification of the original software or it can be the basis of a new software project.

In addition to modification, another common practice with open source software is the reuse of unmodified open source software to build larger and more complex systems. This is enabled by a combination of the modularity as previously described as well as the licensing and cultural norms around open source software. While such reuse can and does happen with proprietary software, it usually requires that developers acquire potentially expensive licensing to do so. With open source, the largest cost is either the expertise required to build the new software system or the fact that, with some licenses, the resulting software once distributed must also be distributed as open source.

### **6.2.1 Reuse in the Google Chrome Web Browser**

In September of 2008, Google made headlines by announcing that they had developed and were releasing a new web browser, Google Chrome[Pichai and Upton, 2008]. This browser was largely functional at that point and was to be released as open source and then continue to be worked on and improved by Google engineers in collaboration with other open source developers who wished to contribute.

What is interesting is that rather than develop everything for the browser themselves, they decided to reuse a large number of existing open source components within their browser and only replace the things which they felt “needed” to be redone. Looking at the initial release of the Chrome source code, they were reusing at least twenty-five existing open source libraries[Krumins, 2008]. While some of these were other projects developed within Google such as the v8 JavaScript engine and the Breakpad crash reporting system, some of the reuse was of components which are notable large parts of competing open source browsers. These include the Netscape Portable Runtime (NSPR) and Netscape Security Services (NSS) initially developed

by Netscape for use in Netscape Navigator and still in use today with the Firefox web browser and also the open source Webkit HTML rendering engine which is used by the Safari web browser.

Without such reuse, Google would have needed to develop all of these components themselves. Instead, they have built their browser on top of them and even contributed changes back to the initial projects as a result of their use within Chrome.

## 6.3 Reuse in Web Services

Reuse in web services can also take on several different characteristics.

The first of these is the reuse of other web services. While not intrinsic to web services, the simple architectural nature tends to lead to a desire to reuse the services. There also is a large cultural aspect among web service developers that encourages such reuse which is rooted in the idea of a mashup. According to the Open Mashup Alliance[Open Mashup Alliance, 2009] a mashup is a combination of other existing data sources, including web services, to produce more interesting data.

Secondly, web services themselves are frequently able to take advantage of a small loophole in open source licensing often referred to as the ASP loophole[FabrizioCapobianco, 2006]. As the restrictions of most open source licenses only take effect when the software is distributed, web services can generally reuse open source components without having to be released themselves as open source. Thus a huge number of web services reuse, modify and improve on existing open source software but do not necessarily contribute back any of their modifications. Also, and perhaps more importantly, they do not in turn force their own software to be open source.

The growing use of this loophole has been met, with the creation of new licenses such as the Affero GPL[AGPL] which places strong restrictions around the requirements for the distribution of changes even in cases where there is no actual software distribution taking place and instead users are interacting with a web service. AGPL

adoption rate is still quite slow compared to the overall growth of both web services and open source, being measured at a rate of only fifteen projects per month[Kuhn, 2008] in December of 2008 and with only a total of 229 projects having adopted the license overall as of November 2009[Black Duck].

That said, if AGPL adoption were to begin to pick up at a significantly greater pace, it would lead to a need for change by web services providers either so that their services would be open sourced as well or to no longer reuse the open source components that they can today.

### 6.3.1 Reuse of Facebook Web Services

Facebook is a social networking site that was founded in 2004, originally to help college students connect with others they went to college with. Since then, it has grown into a site allowing anyone to join and with a massive global appeal, seeing nearly 100 million visits per months as of late 2009[Eldon, 2009]. The service is comprised of a number of features from real-time communication and offline messaging to a platform that others build games and other applications on top of. Virtually all of these are developed either as web services or on top of the web services provided as the “Facebook platform”.

The first way that Facebook allows for reuse is by having relatively easy to use web services[Facebook] to have a developer’s own application appear as a Facebook application and thus usable and viewable by any Facebook user. Part of this integration allows for things such as the addition of comments by other Facebook users who may not use your own application. These comments can then be pulled back to your own application using the *Comments.get* web service call.

This has allowed other web applications for a variety of other users to be present within Facebook. For example, this gets used by fitness related applications to let their users share information via Facebook. One example is the Motionbased.com site which allows a user to publish any fitness activities that they have tracked with a

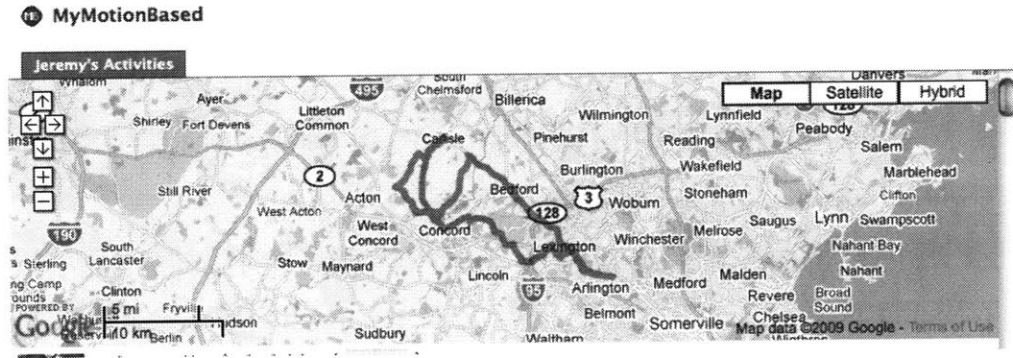


Figure 6-1: The integration of MotionBased within Facebook, Facebook.com, 24 November 2009

GPS to their Facebook profile and then pull information from that posting back into Motionbased site (Figure 6-1). A second example that works similarly but with less of an interactive nature is the MapMyRide.com application[MapMyRide.com, 2009]; this application allows a cyclist to upload their ride information to MapMyRide.com and then reuse that data on Facebook.

This ability to easily embed and reuse components via web services calls is a delivery on some of the promises which were made with object oriented programming, just fifteen to twenty years later and via a substantially different programming model.

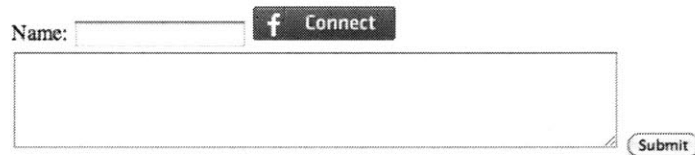
Facebook also encourages the reuse of some of their components completely independent of the use or integration with Facebook itself. An example of this is their Facebook Connect service[Facebook]. As the usage of the Internet and various web sites has grown, people have frequently needed to provide a log in to websites to store information or be recognized on future visits. Reasons for this are from simple things such as remembering layout choices which you have made for a site to remembering your identity when contributing content in the way of comments or reviews to a complete knowledge of who you are on e-commerce sites to be able to allow you to purchase items without entering your credit card information every time.

Facebook Connect provides a simple API that a website owner can use to allow their users to log in with their Facebook username and password rather than having

a new identity. This frees website owners from having to build complex systems and worry about the security of their system and instead allows them to just put in simple references to use existing JavaScript and then the HTML tag `<fb:login-button />`. A full example web page making use of this can be found in Appendix A.

This web page is a very simple comment form, but is illustrative of how a more complex site could easily make use of the Facebook connect API and how allows a user to log in to your site using Facebook by clicking the *Connect* button (Figure 6-2), authorizing the site (Figure 6-3) and then being logged in (Figure 6-4).

## Give me feedback




Name:  

Figure 6-2: Before logging in, the user sees a *Connect* button



Figure 6-3: The user allows the site to use their Facebook identity

### 6.3.2 Facebook's Reuse of Open Source

In addition to allowing for these types of reuse of their own components, Facebook has been a strong user of open source software in the building of their own software

## Give me feedback



The screenshot shows a web form titled "Give me feedback". At the top, there is a small profile picture of a person and the text "Name: Jeremy Katz". Below this is a large, empty rectangular text input field. To the right of the input field is a rounded rectangular button labeled "Submit".

Figure 6-4: The site can then show the user's Facebook identity and use it

stack. They make heavy use of open source components such as Linux, Apache, MySQL, PHP, and Memcached[Agarwal, 2008]. Therefore, without these open source components, they would have had a substantially more difficult time building and growing the service. While they could be doing so without any contributions back to the greater open source community, Facebook is actually considered to be a “good citizen” and extensively does attempt to give back their modifications to open source software that they themselves use.

This giving back, though, does not extend to the software which makes up Facebook itself. If you wanted to build your own Facebook, you would only have the building blocks and not the entirety of the code which makes up the site. One could argue that in this case, the code itself isn't even the interesting or important part but instead the data which is contained within Facebook is more necessary.

# Chapter 7

## Economics

While the technical aspects of open source and web services are interesting, each also faces significant challenges when it comes to actually using the technologies in a business to make money.

Looking at the software industry as a whole, there have historically been two primary ways in which software companies have been able to make money. The first of these is as product companies, in which the software company develops software and then sells it. The second is services companies where the software exists to help in the sale of complementary services. [Cusumano, 2004]

In the worlds of open source software and web services, we can find elements of each of these but there are also unique twists and problems encountered by each.

### 7.1 Making Money with Open Source

Making money with open source software is a concept that seems counter to some of the primary ideas behind such software. The idea of a community of contribution and allowing others access to not just modify but also *redistribute* the source code to your software means that the typical method of charging for software becomes much more difficult. Generally, you would not be able to have a copy of the software until

you pay a licensing fee to the author and distributor of the software. In open source, a user can instead always get access to the software and even install and execute the software. This challenge is perhaps why there are so few successful open source software companies.

### **7.1.1 Making Money as an Open Source Software Company**

As previously mentioned, one benefit of open source can be the symbiotic relationship which can exist between a company and the open source software that they distribute. This is mostly seen in software product companies which are focused on building a software product to distribute and sell to customers. Looking at the set of these companies, the biggest success is probably Red Hat. Red Hat is a pure software company which builds and distributes several successful open source products including:

- The Red Hat Enterprise Linux operating system is a certified and tested build of a variety of open source components.
- The JBoss Enterprise Middleware Suite is a certified stack of components necessary to build internal business-critical applications

All of the components of both Red Hat Enterprise Linux and JBoss are open source and thus available to anyone in the world who wishes to download, compile and assemble them on their own. And in fact, in the case of Red Hat Enterprise Linux, one can actually download such a compilation and run it in the form of CentOS, a rebuild of the Red Hat Enterprise Linux components with the trademarks removed.

But Red Hat has managed to make nearly \$650 million in revenue in their last fiscal year, \$550 million from the sales of software[rht, 2009]. This is because businesses are willing to pay Red Hat to receive a copy of the software which has been certified by Red Hat as being extensively tested and supported to a standard meeting their needs according to the results of that testing. They also pay for a version of the software



which is certified by any of a number of independent software and hardware vendors to play well with this third party product.

As this software is available to anyone to download, Red Hat also has taken the stance of providing the software on a “subscription” basis. That is, a customer purchases access to copies of the certified software and the support for a period of time and continues to pay on a recurring basis for that access. Given that new versions of the software would be available to anyone, this subscription includes the idea of not being tied to any specific version of the software that is distributed by Red Hat and instead provides access to all versions. While not a concept which is unique to open source, the angle of doing it as a result of the availability of all versions of the software is and it is one which has been somewhat useful to Red Hat in encouraging adoption of newer versions of their software.

It is unclear whether this approach, though, is one which can be easily replicated by others. [Vance, 2009] Other software companies such as Alfresco, which provides an open source content management system, and SugarCRM, which provides an open source customer relationship management system, do not seem to have the same level of success as Red Hat. It is possible that this can be attributed to their more limited time being available for customers to purchase. But it is equally possible that there is something different about these other businesses where the market which they are selling into is one which is less technical and where there is less desire to become an expert in the system.

### **7.1.2 Making Money with Open Source as a Complement**

That said, there are now a number of other companies which are able to make money more as a services company as a result of their open source software contributions and distribution, although it is not the primary source of their revenue. In many cases, it is not even selling software which leads to the revenue stream for these complement companies. But the expertise gained in the software through being involved in its

development places them in a unique position to provide value to their customers in a services role.

IBM is perhaps the largest of these companies. In 2000, IBM committed to the idea of spending one billion dollars on open source software[Lock, 2002]. Now, nearly a decade later, they are one of the largest contributors to the world of open source with contributions ranging from low-level work such as the Linux kernel to a variety of web infrastructure projects under the Apache umbrella to being one of the largest contributors of code to the Eclipse Integrated Development Environment. Even with all of these contributions, though, the amount of revenue that they get as a direct result of this distribution is relatively small. The vast majority of their revenue continues to come from a combination of their hardware and services businesses; open source serves purely as a complement helping to drive customers to purchase more IBM services and hardware.

## 7.2 Making Money with Web Services

Web services face some challenges as well when it comes to making money. This isn't, though, generally a result of the easy availability of the software.

One of the big challenges faced by web services is that their costs are not, as with most software, purely based on the cost to develop the software. Instead, there is a large cost associated with actually hosting and providing the software to its users. This operational cost can range from extremely small if there are few users to a very large amount if the number of users is large. In addition to the operational costs for hosting the software, one thing which is extremely important in these web services is the data which goes along with it. Collecting, retaining, backing up and all of the other operations needed on a day to day basis to ensure that your customers do not lose data can also be a huge cost.

Therefore, to make money with web services, a provider has to have some way

to make up for these operational costs in addition to recouping the costs of the development of the software itself.

### **7.2.1 Making Money with Consumer Web Services**

Consumer web services face a huge challenge when it comes to monetization. Consumers are at this point largely accustomed to the idea of not paying for things which they access and use on the Internet. While there are exceptions of services which can effectively charge for access, even these rare cases seeing decreasing acceptance and usage as the idea that the Internet means free continues to take hold.

The only way that really seems to be effective for monetizing these services at this point tends to be advertising included along with the service. Advertising as a way of making money goes back to the early days of the World Wide Web when banner ads were displayed on pages and users “clicking through” to view the destination of the ad would in turn provide some small amount of revenue to the operator of the original web site. The value in these has decreased as users have become more accustomed to them and find that they are not relevant to what they care about.

Advertising in web services today, therefore, takes a far more targeted approach. The data about how you use the web service is used to help increase the applicability of the ad and thus increase the value of its display. One example of this is the advertising within the Google search engine; Google allows their customers to bid and pay for ads to show up when people search for certain keywords. This allows you to know that your ad as a fireplace company will show up when people are searching for information on fireplaces. This sort of targeted advertising is far more effective at attracting customers and thus much more valuable.

### **7.2.2 Making Money with Business Web Services**

With businesses, this idea of everything on the Internet being free is less prevalent and companies are, in fact, willing to spend money for things which they see as providing

them value.

One of the bigger concerns for a business in paying for use of a web service is that there is little concrete which you actually receive for your purchase. Since the software you are using is not directly running on hardware that is physically owned by the company, what would happen if the company you had paid a fixed rate were to go out of business and thus no longer be able to provide you with their service? Given this concern, much as Red Hat has done with open source software, most business web services are purchased on a subscription basis. This allows companies to have access to the service for a discrete chunk of time and then pay again to continue to have access for the next discrete chunk.

This has provided a couple of benefits for the purchasers of these web services. The first is that there is less liability around what happens if a service were to go away. Given that you have only paid for a short period of time, the chances are that you have gotten to use the service for a significant chunk of that time. The other benefit is that you are able to transform these software purchases from having to be recorded as capital expenditures in your financial statements and instead can record it as an operational expenditure. This has some advantages in making your financial statements less complex as well as not having to worry about any depreciation over time for the software as the usable lifetime of the software is just what you have paid for up until that point.

# Chapter 8

## Conclusion

While there are a small number of open source web services currently being developed and in use, the growth there has not mirrored that of the growth of open source in general. Even companies which are focused around open source technologies have tended to build far less open web services. This has been seen everywhere even including Red Hat with their Red Hat Network service for systems management which was only opened after many years of requests and criticism[Red Hat, Inc.].

While open source provides a set of advantages in terms of the architecture of the system developed, many of the more prominent of these are also present in web services as an emergent property for other reasons.

Modularity is a strong focus of open source development as a result of the distributed nature of the development teams. Web services also achieve a high degree of modularity, but rather than it being based on the distribution of the developers working on the software, it is instead based on the way in which the software is run on a distributed network of computers providing the services. Modularity also occurs as a result of the defined standards for web services and the requirements for clear interfaces between the components.

Web services also are able to follow with some of the strongest tenets of bazaar development through frequent releases. There may even be an easier path to achieving

this goal with web services than with open source software as one does not have to actually deliver any software to a user that then has to be installed and configured. By having these frequent releases, web services are able to iterate and meet the needs of their users.

The combining and reuse of web services via mashups is also very similar to the reuse of open source components. There is a strong culture of working with existing services rather than inventing your own wheel. Since one does not necessarily have to pay expensive licensing fees to use data from another web service also helps to encourage this reuse. As the licenses of most open source software only require the release of modifications or your own code on a distribution event of your software, web services are exempt from this reciprocal nature of open source licensing and can also reuse substantial amounts of open source software without themselves having to be released as open source.

And finally, given the relative difficulty in making money off of web services, developers of such services are unwilling to give up anything which could be an advantage towards doing so. If that means that the source code of the service remains proprietary and they cannot share the development costs, then that is the approach that is taken. Given that the majority of the cost for the services is not the development cost and is rather the operational expenditures to run the service, releasing the software as open source and reducing the development cost is of little benefit.

Given all of this, the growth of open source in web services seems unlikely without another more significant change occurring. That said, the growth of a licensing model such as the Affero GPL as mentioned in Section 6.3 could lead to a growing number of open source web services due to a requirement of either open sourcing or no longer being able to reuse other open source components.

The other area of web services that is seeing some open source growth is as an exit strategy. When a web service is bought or has to shut down for some reason, there is perhaps the beginning of a trend to release the source code which is used for running

the service as open source. Most recently, this was seen with Google's acquisition and shut down of the Etherpad service in December of 2009. Due to the outcry from the existing user-base, the source code behind Etherpad has been released as an open source project[Iba, 2009].





# Appendix A

## Facebook Connect Sample Code

The HTML source code in the listings below gives an example web page making use of the Facebook Connect[Facebook] API to provide a way for a user to log in to a web site. This is the trivial example as used for Figures 6-2, 6-3 and 6-4. Two files are required for this functionality.

- *index.html* provides most of the necessary functionality and all of the displayed content for the page (Listing A-1)
- *xd\_receiver.html* is a simple file which is looked at by the server side of Facebook to ensure that you are intending to allow users to log in via the Facebook Connect API (Listing A-2)

In a non-trivial example, this type of code could be integrated as a part of a larger web site which allowed a user to comment on articles or any other action which requires a user login. One example that does this is CNN.com where they allow logging in for discussion of their news articles (Figure A-1).



Figure A-1: Using Facebook Connect on CNN.com, 24 November 2009

Listing:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
  xmlns:fb="http://www.facebook.com/2008/fbml">
<body>
<form>
  <div id="userdiv">
    Name: <input name="name">
    <fb:login-button onlogin="login();"></fb:login-button>
  </div>
  <textarea name="message" rows=5 cols=80></textarea>
  <input type="submit" value="Submit">
</form>
<script type="text/javascript">
function login() {
  var u = document.getElementById("userdiv");
  u.innerHTML = "Name: <fb:name uid=loggedinuser
  useyou=false />";
  u.innerHTML += "<fb:profile-pic uid=loggedinuser
  facebook-logo=true />";
  FB.XFBML.Host.parseDomTree();
}
</script>
<script type="text/javascript"
  src="http://static.ak.connect.facebook.com/js/\
  api_lib/v0.4/FeatureLoader.js.php">
</script>
<script type="text/javascript">
  FB.init("APIKEY", "xd_receiver.html");
</script>
</body>
</html>
```

Listing A-1: Sample source code for interacting with Facebook Connect

Listing:

```
<html>
<body>
<script
  src="http://static.ak.connect.facebook.com/js/\
    api_lib/v0.4/XdCommReceiver.js"
  type="text/javascript">
</script>
</body>
</html>
```

**Listing A-2:** xd\_receiver.html standard code to provide access between Facebook and your website

# Bibliography

- Alleyoop valgrind frontend. World Wide Web, March 2009. URL <http://alleyoop.sourceforge.net/>.
- Bluefish editor. World Wide Web, October 2009. URL <http://bluefish.openoffice.nl/>.
- git: The fast version control system. World Wide Web. URL <http://git-scm.com>.
- Red hat, inc 2009 form 10-k. World Wide Web, 2009. URL <http://www.sec.gov/Archives/edgar/data/1087423/000119312509091983/d10k.htm>.
- Aditya Agarwal. Facebook: Science and the social graph. In *QCon San Francisco 2008*, 2008. URL <http://www.infoq.com/presentations/Facebook-Software-Stack>.
- John Allspaw and Paul Hammond. 10 deploys per day. In *Velocity 2009*, 2009.
- Amazon. Amazon web services. World Wide Web, 2009. URL <http://aws.amazon.com>.
- Carliss Y. Baldwin and Kim B. Clark. Managing in an age of modularity. *Harvard Business Review*, 75(5):84 – 93, 1997. ISSN 00178012.
- Tim Berners-Lee. The world wide web: Past, present and future. World Wide Web, August 1996. URL <http://www.w3.org/People/Berners-Lee/1996/ppf.html>.
- Black Duck. Open source license data. World Wide Web. URL <http://www.blackducksoftware.com/oss/licenses/#adoption>.
- David Booth, Hugo Haas, Francis McCabe, Eric Newcomer, Michael Champion, Chris Ferris, and David Orchard. Web services architecture. World Wide Web, February 2004. URL <http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/>.
- Martin Campbell-Kelly. Historical reflections will the future of software be open source?. *Communications of the ACM*, 51(10):21 – 23, October 2008. ISSN 00010782.

- Piero Colagrosso. Formal specification of c++ class interfaces for software reuse. Master's thesis, Concordia University (Canada), 1993.
- Computer History Museum. Computer history museum. World Wide Web. URL <http://www.computerhistory.org/timeline/?year=1964>.
- Jon Corbet. Kernel summit: Development process. World Wide Web, July 2004. URL <http://lwn.net/Articles/94386/>.
- Michael A. Cusumano. *The Business of Software: What Every Manager, Programmer and Entrepreneur Must Know to Thrive and Survive in Good Times and Bad*. Free Press, 2004.
- Kevin C. Desouza, Yukika Awazu, and Amrit Tiwana. Four dynamics for bringing use back into software reuse. *Commun. ACM*, 49(1):96–100, 2006. ISSN 0001-0782. doi: <http://doi.acm.org/10.1145/1107458.1107461>.
- Eric Eldon. Compete, comscore and quantcast show facebook us october traffic up, myspace and twitter traffic down. World Wide Web, November 2009. URL <http://www.insidefacebook.com/2009/11/13/compete-comscore-and-quantcast-show-facebook-us-october-traffic-up-myspace-and->
- Mark Endrei, Jenny Ang, Ali Arsanjani, Sook Chua, Philippe Comte, Pal Krogdahl, Min Luo, and Tony Newling. *Patterns: Service-Oriented Architectures and Web Services*, page 21. IBM International Technical Support Organization, first edition, April 2004.
- FabrizioCapobianco. The honest public license. World Wide Web, August 2006. URL <http://www.funambol.com/blog/capo/2006/08/honest-public-license.html>.
- Facebook. Facebook developers api. World Wide Web. URL <http://wiki.developers.facebook.com/index.php/API>.
- Facebook. Build and grow with facebook connect. World Wide Web. URL <http://developers.facebook.com/connect.php>.
- Joseph Feller, Brian Fitzgerald, Scott A. Hissam, and Karim R. Lakhani, editors. *Perspectives on Free and Open Source Software*, page 100. The MIT Press, 2005.
- Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, 2000.
- William B. Frakes and Sadahiro Isoda. Success factors of systematic reuse. *IEEE Software*, 11(5):14–19, 1994. ISSN 0740-7459. doi: <http://doi.ieeecomputersociety.org/10.1109/52.311045>.

- Free Software Foundation (FSF). Gnu affero general public license. World Wide Web, June 2009a. URL <http://www.gnu.org/licenses/agpl.html>.
- Free Software Foundation (FSF). Gnu general public license. World Wide Web, June 2009b. URL <http://www.gnu.org/licenses/gpl.html>.
- Erich Gamma, Richard Helms, Ralph Johnson, and John M. Vlissides. *Design Patterns*. Addison-Wesley Professional, 1994.
- Rochelle Garner. Salesforce.com welcomes winter with latest release. <http://www.crn.com/it-channel/18831164;jsessionid=4AL5IQSG5D513QE1GHRSKH4ATMY32JVN>, December 2003.
- Stefan Haefliger, Georg von Krogh, and Sebastian Spaeth. Code reuse in open source software. *Management Science*, 54(1):180–193, 2008. ISSN 00251909. URL <http://www.jstor.org/stable/20122369>.
- Aaron Iba. Etherpad open source release. World Wide Web, December 2009. URL <http://etherpad.com/ep/blog/posts/etherpad-open-source-release>.
- Kernel.org. Linux kernel release dates. FTP. URL <ftp://ftp.kernel.org/pub/linux/kernel>.
- Peteris Kruminis. Code reuse in google chrome browser. World Wide Web, September 2008. URL <http://www.catonmat.net/blog/code-reuse-in-google-chrome-browser/>.
- Bradley Kuhn. Agpl declared dfsg-free. World Wide Web, December 2008. URL <http://autonomo.us/2008/12/agpl-dfsg-free/>.
- Florian Lindner. Service oriented architecture elements. World Wide Web. URL [http://commons.wikimedia.org/wiki/File:SOA\\_Elements.png](http://commons.wikimedia.org/wiki/File:SOA_Elements.png).
- Tony Lock. Ibm’s billion dollar linux gamble pays off. World Wide Web, 2002. URL <http://www.it-director.com/content.php?cid=2521>.
- Alan MacCormack, John Rusnak, and Carliss Y Baldwin. Exploring the structure of complex software designs: An empirical study of open source and proprietary code. *Management Science*, 52(7):1015–1030, July 2006.
- MapMyRide.com. Mapmyride facebook application. World Wide Web, 2009. URL <http://apps.facebook.com/mapmyride>.
- M. Muffato and M. Roveda. Developing product platforms: analysis of the development process. *Technovation*, 20(11):617–630, November 2000.

- Netcraft Ltd. November 2009 web server survey. World Wide Web, November 2009. URL [http://news.netcraft.com/archives/2009/11/10/november\\_2009\\_web\\_server\\_survey.html](http://news.netcraft.com/archives/2009/11/10/november_2009_web_server_survey.html).
- OASIS. Reference model for service oriented architecture. World Wide Web, October 2006. URL <http://docs.oasis-open.org/soa-rm/v1.0/soa-rm.pdf>.
- Stephen O'Grady. Amazon: Perfect is the enemy of good. World Wide Web, May 2009. URL <http://redmonk.com/sogrady/2009/05/22/amazon-perfect-is-the-enemy-of-good/>.
- Open Mashup Alliance. Open mashup alliance frequently asked questions. World Wide Web, 2009. URL <http://www.openmashup.org/faq/#1>.
- Tim O'Reilly. What is web 2.0: Design patterns and business models for the next generation of software. World Wide Web, September 2005. URL <http://oreilly.com/web2/archive/what-is-web-20.html>.
- Open Source Initiative (OSI). The open source definition. World Wide Web, 2009a. URL <http://opensource.org/docs/osd>.
- Open Source Initiative (OSI). History of the OSI. World Wide Web, 2009b. URL <http://www.opensource.org/history>.
- Sundar Pichai and Linus Upson. Google blogs: A fresh take on the browser. World Wide Web, September 2008. URL <http://googleblog.blogspot.com/2008/09/fresh-take-on-browser.html>.
- Thomas E. Potok, Mladen Vouk, and Andy Rindos. Productivity analysis of object-oriented software developed in a commercial environment. *Software: Practice and Experience*, 29(10):833–847, 1999.
- Eric S. Raymond. *The Cathedral and the Bazaar*. O'Reilly Media, October 1999a.
- Eric S. Raymond. *The Cathedral and the Bazaar*, page FIXME. In Raymond [1999a], October 1999b.
- Red Hat, Inc. Spacewalk frequently asked questions. World Wide Web. URL <http://www.redhat.com/spacewalk/faq.html#whynow>.
- Refsnes Data. Browser statistics. World Wide Web, 2009. URL [http://www.w3schools.com/browsers/browsers\\_stats.asp](http://www.w3schools.com/browsers/browsers_stats.asp).
- Katrina Rhoads Lindholm. The user experience of software-as-a-service applications. Technical report, UC Berkeley: School of Information, 2007.



Salesforce.com. Salesforce.com. World Wide Web, 2009a. URL <http://www.salesforce.com/company/>.

Salesforce.com. Salesforce.com programmable user interface. World Wide Web, 2009b. URL <http://www.salesforce.com/platform/cloud-platform/programmable-ui.jsp>.

Richard Stallman. About the gnu project. World Wide Web, January 2008. URL <http://www.gnu.org/gnu/thegnuproject.html>.

Terry Sweeney. It executives take a more modular approach to crm software. World Wide Web, May 2003. URL [http://articles.techrepublic.com.com/5100-10878\\_11-5034686.html](http://articles.techrepublic.com.com/5100-10878_11-5034686.html).

Karl T. Ulrich and Steven D. Eppinger. *Product Design and Development*, page 165. McGraw-Hill International, fourth international edition, 2008.

Ashlee Vance. Open source as a model for business is elusive. World Wide Web, November 2009. URL [http://www.nytimes.com/2009/11/30/technology/business-computing/30open.html?\\_r=2&ref=business](http://www.nytimes.com/2009/11/30/technology/business-computing/30open.html?_r=2&ref=business).