
An Overview of Open Source Software and Methods of Optimizing the Information Architecture of an OSS Data Repository

Ryan M. Milligan
University of Notre Dame
Undergraduate Student – Dept. of Computer Science and Engineering

Advisors: Dr. Greg Madey and Matthew Van Antwerp
Submitted December 2007

Introduction

Ever since Microsoft emerged as a leader in computer applications development, users have insisted on purchasing proprietary software titles. Entire operating systems complete with word processors, industry-based design tools, and entertainment programs are available for a price, and for the most part suffice in terms of performance. What many consumers are not aware of, however, is that proprietary packages are not the only ones available for public use.

“Open source” software has opened up a new realm of freedom to developers and users alike, and looks to compete with its proprietary counterparts. The first half of this paper will present some of the basic concepts, examples, and issues of open source software. The second half focuses on Zerlot, Notre Dame’s research-based open source data repository, and how it can be modified so that information is easier to find.

Scope

This research paper was written as a component of credit-based undergraduate research at the University of Notre Dame. The research was advised by Dr. Greg Madey of the Computer Science and Engineering department.

Defining Open Source

Written and released by Bruce Perens in 1998, the Open Source Definition outlines the requirements that a piece of software must abide by to be considered open source. Some criteria include the following: ^[1]

- *Free Redistribution*: There can not be any restrictions from selling or giving away the software.
- *Source Code*: All users must have access to the source code (or a means of obtaining it), as well as a compiled binary form.
- *Derived Works*: Developers must allow for future modifications that can be distributed with the same terms.
- *Distribution of Licenses*: Any rights that are described in an open source license must apply to all users.

There are often misconceptions when open source software is referred to as being “free”. This should not be interpreted as free of cost; developers can still charge distribution fees. Instead, one should think of it with respect to the freedom of modifying and re-releasing a currently existing piece of software. This naturally allows for a significantly larger community of engineers and developers, which in many ways can serve as an advantage over proprietary companies.

Early Movements

Much of what open source software is today is due to the work of Richard Stallman, a software developer who began working at the MIT Artificial Intelligence Laboratories in 1971. At the time, terms such as “free software” and “open software” had not been coined yet, but the same concepts were being used. After the PDP-10 series of computers was discontinued, newer models required disclosure agreements from the users. This led Stallman to formulate some assumptions about proprietary software publishers:^[2]

- They think they have the natural right to own software and thus have power over the users.
- They think the only important aspect of software is what functions it allows users to perform.
- They think there would not be any usable software if companies were not offered power in return for their development efforts.

While the criticisms were harsh, Stallman did have a valid complaint, and one which was shared by his colleagues. They considered joining a proprietary company as being a misuse of their talents, while straying away from the technical discipline altogether would have been a waste. Facing a limited number of options, Stallman decided to take his own route.

He began by writing an operating system that he named GNU, a recursive acronym for “GNU’s Not Unix”. In addition, he worked on a variety of applications that even today can be found on nearly every development machine, namely the Emacs text editor and the GCC compiler for building binary versions of code.^[2] While working, Stallman made it a point that he wanted his products to be “free” for public use. In other words, users could run the program, modify it to fit their own needs, and redistribute it with their own modifications

included. This led to a concept known as “copyleft” which, as its name suggests, is meant to act in a way opposite of a copyright. While a copyright is responsible for adding restrictions in an attempt to protect the work of a developer or development group, a copyleft forbids those restrictions from being added to a piece of software. This keeps the code openly available and subject to any modifications users see fit.

Ideas such as these eventually led up to the establishment of the Free Software Foundation in 1985.^[2] The FSF currently serves as a tax-exempt charity for supporting the growth of open source software, and gets most of its income from various services.

Emergence of Linux

In the early 1990s, a Finnish software engineer by the name of Linus Torvalds revolutionized the computing world with his development of the Linux operating system. Now widely used around the world for both personal and professional use, Linux was built from scratch to resolve porting issues that posed a problem at the time.

Initially, Linux was not meant to be widely portable; the main focus was simply to have an operating system with a solid, reliable design. Contrary to some beliefs, no code was borrowed or reference from Unix systems.^[3] Torvalds took a “monolithic” approach, dividing memory into both a kernel space (for core functionality) and a user space. While such an approach is risky in case one wants to add new code or features, the decision ultimately paid off in the long run. The popularity of Linux grew gradually at first, then sky-rocketed once others began to realize its value and make their own contributions. While the kernel has for the most part remained in tact, numerous distributions of Linux have been released since its initial debut. The following statistics give some insight into its rapid growth:^[3]

Lines of Code in the Linux Kernel	
Month / Year	Lines of Code
September 1991	10,239
March 1994	176,250
March 1995	310,950
January 1999	1,800,847
January 2001	3,377,902
December 2003	5,929,913

The rapid growth of the Linux kernel is largely due to the increased number of developers who dedicate their time and effort to making modifications and additions, which is made possible with the principles of open source. With tools such as the aforementioned Emacs text editor and GCC compiler at the disposal of users, the Linux operating system itself has managed to put up a respectable fight with proprietary competitors. While the user and developer communities continue to grow, so will the understanding of how robust and powerful open source software can be.

Open Source versus Proprietary

One common misconception is that today's open source software developers differ from proprietary developers with respect to ideologies and techniques. This is far from the case; in fact, the two groups often learn from each other. The majority of differences are found within the programs themselves, which are built with similar tools in similar environments.^[4]

Open source benefits from the fact that generic code libraries are readily available for use. Proprietary companies have their own libraries, but are limited since the public is not permitted to make additions or improvements to them.

One of the keys to success in any industry is communication, which at times may prove to be a struggle in the open source world. With so many contributors working on a variety of different projects, it is a

challenge to keep everyone on pace, whereas proprietary companies keep it condensed.

Finally, distribution and marketing plays a significant part in the eyes of consumers.^[4] Large corporations such as Microsoft have a clear advantage over competitors when it comes to exposure. Open source developers, however, are using new approaches, such as making their products available online instead of having to pay for CDs to put software on. The Linux installation procedure, for example, is still a work in progress but moving in the right direction.

Development Processes

While there is an abundance of open source products available for use, each one undergoes a generic development process that has the following characteristics:^[5]

- *Parallel development:* People work on different aspects of the project at the same time, requiring solid communication between team members.
- *Large communities:* In fact, open source software projects often go so far as to be international in scope.
- *Independent peer review:* With such a large developer base, the odds of spotting a bug in the code increases, which is not something that is guaranteed in traditional proprietary environments.
- *Developer participation:* Large-scale open source projects tend to be effective in attracting more skilled developers from around the world.
- *Rapid releases:* The consumer's focus is always on the product, so to keep up with competitors, the time-to-market must be kept as short as possible.

Implementation, not planning, is typically the key phase in open source development. As a

result, code may be released before it is entirely debugged, then re-released once the problems are attended to.^[5] Compromises such as this are made to ensure that the project is moved along at a pace that allows it to keep up with competition.

Stakeholders

While freely redistributable, it should still be noted that many people rely on a steady open source development stream. Some of these groups include:^[5]

- *Developer Communities*: Studies by Linux show significant diversity within open source groups in terms of gender and ethnicity, but many are specialists and professionals who have viewed open source software as a career path.
- *User Communities*: Demographic statistics are hard to come by, but those that exist also show noticeable trends.
- *Commercial Organizations*: As an example, IBM builds and sells complete Linux systems as an independent software vendor. Companies like Red Hat can also be characterized as value added resellers.
- *Non-Commercial Organizations*: These include regular clients of open source products, such as Apache, GNOME, and Python.

What follows in the next section are descriptions of some of the open source products that help these communities conduct research or business.

Examples of OSS

Internet-Based Tools

The Mozilla Project was launched in the earlier part of 1998 when Netscape decided to make some of its source code

available to the public.^[6] A Mozilla Public License was written and later approved by the Open Source Initiative. The community consisted mostly of Netscape employees, but the development style gradually shifted from a proprietary to an open source model. A public bug-tracking system was set up, and the Mozilla Foundation was established in 2003 as a nonprofit organization. When v1.0 of Firefox and Thunderbird was released one year later, over two million downloads were recorded in the first two days.^[6]

Development Environments

Cygwin is a tool that has been widely accepted by those who own a Windows machine but prefer the terminal interface of Linux. It consists of a data link layer which acts as a Linux API emulation, as well as a collection of tools which provide a Linux look and feel, such as the GCC compiler.^[7] It is currently maintained by employees of Red Hat, and is freely redistributable under the terms of the GNU Public License (to be discussed in detail later in this report).

File-Sharing Systems

One of the most complex open source projects in development is Samba, a software suite that provides file services to both Windows and Linux clients with the intent of creating a “glue” between the two.^[8] While very useful, it is one of the more difficult projects to join, because the candidate has to have a thorough knowledge of both the Win32 and POSIX APIs in order to make meaningful contributions. As with Cygwin, Samba is available as open source software under the GNU Public License.

Versioning Systems

Versioning systems are used widely throughout industry as a method of tracking

earlier releases of software or documentation and running source code comparisons as the code evolves over time. A number of open source tools are available; some of the more popular examples include Subversion (used by the Apache Software Foundation and released under the Apache License), BitKeeper (formerly – the company changed to a proprietary version in 2005), and Git (created by Linus Torvalds and used directly with the Linux kernel).

Mathematical Applications

Software has greatly contributed to mathematical-based research. In addition to the impact that products such as Linux and Mozilla have had on computing, applications such as TeX have made things easier for universities and corporations alike.^[9]

Something interesting to take note of in this situation is that mathematical theorems, once formulated and confirmed, require very little maintenance. Mathematical software however, can potentially be very costly with respect to the man-hours needed to fix a bug or write the software in another language. This is where the size of the open source community benefits its stakeholders; having more developers will often result in less time required to make updates.

Scripting Languages

Scripting languages serve a wide variety of purposes; while they are not initially intended for full-fledged application development, they can be very useful for automation and dynamic web page content. Some examples of scripting languages include Perl, which was developed by Larry Wall, and Python, which was developed by Guido Von Rossum.

Again, the size of the open source community is responsible for the making the languages what they are today.

Other Issues in Open Source

Profitability

With much of the available open source software being distributed at little or no cost, people often ask how it is possible to make a career out of it. One of the best examples of how an open source business model can be profitable is Cygnus, which was founded in 1989.^[10] When the company started up, proprietary software was caught in a struggling state, so Cygnus developed its own concepts of finance, accounting, and marketing. What made it so unique was that engineers were in charge of every aspect, since they knew the products and services that were being sold better than anyone else. Employees focused on servicing the GNU compiler and debugger, and by providing unmatched technical support, had about one million dollars worth of contracts.^[10] Other start-up companies have since tried to implement the same model.

Licensing

There are a number of open source licenses that developers have the option of adopting once a product is ready for release. Each one has its own attributes and is suitable for different scenarios.

One of the most popular licenses is the GNU Public License; of all the major licenses, it is the only one that strictly forbids any form of integration with non-free software. Other options, such as the Lesser GNU Public License and the BSD License, allow mixing to occur.^[1] The variety of choices that are available often results in a higher level of comfort amongst programmers when it comes to contributing to open source; they are assured of the specific rights they have, as well as the rights of those who eventually use the source code for personal or commercial purposes.

Security

The “many eyes” theory states that once a security vulnerability is detected in a piece of code, it is relatively easy to fix as long as there are a number of perspectives and suggestions available. An open source model helps in situations such as these because anyone who wants to take the time and effort to look for potential flaws is able to do so freely. ^[11]

There is actually a growing interest in the claim that open source software may, in some ways, be more secure than its proprietary counterpart. Many users take into account the principle of disclosure; in open source, users are not at the mercy of anyone who “owns” or maintains the code.

A number of projects such as OpenSSL, Enigmail, CVE, and GnuPG are dedicated to ensuring the security of open source software and minimizing the number of potential security flaws in source code. ^[11]

Governance

The rise of the Internet has made the task of discovering possible collaborators and subcontractors for open source projects relatively simple. Complex software is hard to build, so it demands multiple dimensions as well as an even division of labor. ^[12]

While no one is really ever “in charge” of open source software, there are a number of design principles that developers best abide by, including: ^[12]

- *Weighting of Contribution*: Not every developer is equally knowledgeable about a problem, which needs to be taken into consideration.
- *Timing*: Group members need to know how urgently data and information needs to be used, refereed, and updated.
- *Granularity*: Part of being successful, especially with something such as

open source software, is knowing where to draw the boundaries around any given module.

- *Security*: This becomes a greater issue as the value of the system rises over time.

Open Source’s Future

The idea of “open source” has recently extended beyond the scope of software. It is now being embraced by many as a business model – one with various platforms, goals, markets, role players, licenses, and development tools. ^[13]

It is safe to say that open source software has made significant impacts in both the technical world and in the business world. With a solid foundation already set in place, and a user community that is growing every day, it has the opportunity to continue competing with proprietary competitors for position in the markets. Given the oft-ignored ideology of freedom behind open source and the power of today’s proprietary vendors, such an accomplishment would be one that almost goes so far as to defy logic.

Summary

The goal of this part of this research report was to give readers a foundation as to what constitutes open source software and how some of the concepts and ideas are applied throughout the stages of development. A brief history was followed up with examples and issues as a way to compare and contrast open source with proprietary approaches.

The second part of the report shifts the focus from open source software in general to how the University of Notre Dame’s research group analyzes the existing community of OSS users and developers. It is focused on technical work that was done throughout the course of the semester.

About Zerlot

Zerlot is a wiki-based open source project data repository, kept and maintained by Dr. Madey's research group at the University of Notre Dame. The data archives that are available on Zerlot are provided by SourceForge.net, the largest open source project forum on the Internet. Every month, SourceForge sends dumps of data that are organized into various schemas and tables. This way, researchers can visit Zerlot and get continuous insight with regards to trends in the open source community, such as:

- The types of projects being worked on (commercial use, personal use)
- The average number of people working on a single project (which can fluctuate over time)
- How often each project comes out with a new version or release
- The programming languages and platforms that are used by developers

Such an environment is very useful to those who wish to obtain data, so it is important that the information available on the Zerlot repository is easy to find.

Concepts of Information Architecture

Information architecture is typically defined as the science of expressing the model or concept for information. [14] In order to be effective, a shared environment should be designed in such a structure that allows users to access information in a timely and efficient manner. Formally, the IA Institute defines it as: [14]

- The structural design of shared information environments
- The art and science of organizing and labeling web sites, intranets, online communities and software to support usability and findability
- An emerging community of practice focused on bringing principles of

design and architecture to the digital landscape

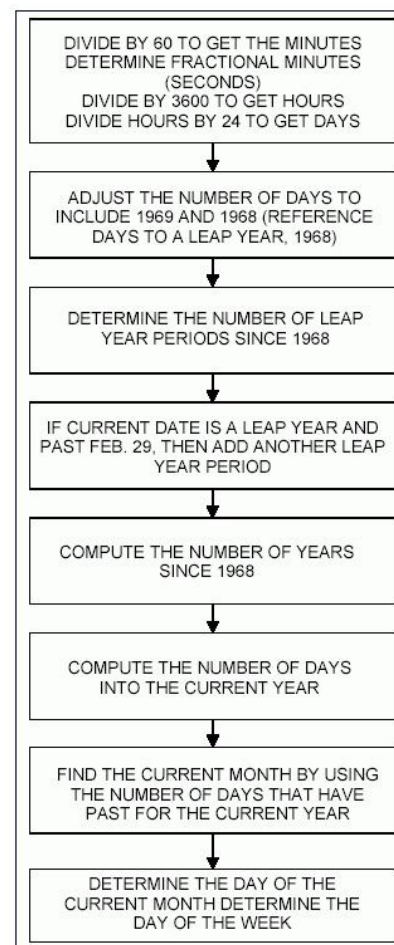
It should be noted that "architecture" in this context is not meant to imply that the information remains static; especially on the Internet, content is dynamic and modified on a frequent basis.

What follows is a description of the technical work that was done throughout the semester in an attempt to improve the current IA of the Zerlot repository.

Technical Work

Unix Epoch Converter

The diagram below illustrates the algorithm of converting a positive integer to its Unix epoch equivalent:



The Unix epoch is defined as the total number of seconds that have elapsed since midnight on January 1, 1970 (a reference set in Unix systems). Time-related data that is received from SourceForge.net is initially shown in this format, so a conversion tool helps users identify standard-format release dates, etc. without having to perform all of the calculations themselves.

A JavaScript conversion tool was originally developed with the appropriate arithmetic statements to get the desired output. However, modifications were made to include built-in functions. Combined with an HTML-based form, users can enter an integer (the timestamp) and convert it to mm/dd/yy @ hh/mm/ss format within a matter of seconds, as seen below.

Timestamp:	Month:	Day:	Year:	Hour:	Minute:	Second:	GMT
<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
<input type="button" value="convert"/>	/	/	@	:	:	:	

The following code is a framework to show how both the function and form are structured:

```
<HTML><HEAD>
<script language="JavaScript">
  function convert()
  {
    var theDate = new Date(...);
    dateStr = theDate.toGMTString();
    arrDateStr = dateStr.split(" ");
    document.form1.numMonth.value =
      getMonthNum(arrDateStr[2]);
    document.form1.numDay.value =
      arrDateStr[1];
    document.form1.numYear.value =
      arrDateStr[3];
    document.form1.numHour.value =
      arrDateStr[4].substr(0,2);
    document.form1.numMinute.value =
      arrDateStr[4].substr(3,2);
    document.form1.numSecond.value =
      arrDateStr[4].substr(6,2);
  }
  function getMonthNum(abbMonth)
  {
    arrMon = new Array("Jan", ...);
    for(i=0; i<arrMon.length; i++)
    {
```

```
      if(abbMonth == arrMon[i])
        return i+1;
    }
  }
</script>

</HEAD><BODY>

<form name=form1>
  <table border=0>
    <tr>
      <th>Timestamp:</th>
      <td valign=bottom rowspan=2>
        <input type=button
          name=switcher value="convert"
          onClick="convert();"></td>
      <th>Month:</th><th>&nbsp;</th>
      <th>Day:</th><th>&nbsp;</th>
      ...
    </tr>
    <tr>
      <td align=center><input
        type=text size=14 maxlength=11
        name=timeStamp></td>
      <td><input type=text size=4
        maxlength=2
        name=numMonth></td><th></th>
      <td><input type=text size=4
        maxlength=2
        name=numDay></td><th></th>
      ...
    </tr>
  </table>
</form>

</BODY></HTML>
```

Table Descriptions Page

Each individual table in the Zerlot schema browser comes with a list of field attributes as well as a space where users can add their own descriptions or comments. While helpful if someone wants to know more about a single table, such an approach may be inconvenient if the user wants to get information from multiple tables at once.

Instead of navigating from table to table, a new page was added to the wiki that consists of nothing but table descriptions and comments, which can be added, modified or removed by users whenever changes are desired. In order to accomplish this, it was

necessary to become familiar with the syntax of the wiki itself.

As a simple example, the following code resembles a generic 3-by-3 table...

```
{| border="2"
|+ SAMPLE WIKI TABLE...
|-
|! Name
|! State
|! Color
|-
|-
| Michael
| Alabama
| Red
|-
|-
| Joe
| California
| Green
|-
|-
| Susan
| Vermont
| Purple
|-
|}
```

...that results in a display that looks like this:

SAMPLE WIKI TABLE...

Name	State	Color
Michael	Alabama	Red
Joe	California	Green
Susan	Vermont	Purple

With a basic knowledge of the wiki syntax, building an actual table of page descriptions was a straightforward task. The sample code was extended to include all instances of tables that appear in at least one schema. The description column, initially void of content, can be filled in by users at their own discretion. As mentioned before, the page itself ultimately provides a more convenient resource for those who wish to find information on multiple tables without having to spend time navigating.

The following code outlines the page:

```
{| border="2"
|+ Through November 2007:
|-
|! Table
|! Description
|-
|-
| *[[activity_log]]
|-
|-
| *[[activity_log_old]]
|-
|-
| *[[activity_log_old_old]]
|-
|-
| *[[activity_log_regs]]
|-
|...
|}
```

Table descriptions

Through November 2007:

Table	Description
*activity_log	
*activity_log_old	
*activity_log_old_old	
*activity_log_regs	
*activity_log_regs_tmp	

Note: The bracket "[[]]" notation in the wiki provides a link to that table's unique page.

Schema Browser Enhancement

At the time this document was written, the schema browser simply lists collections of data, sorted by the month and year of release. When a schema is clicked by the user, a new page opens up with a list of tables that appear in that schema. One of the potential enhancements that were proposed was to edit an existing Perl script in such a way that would turn the tables into HTML-based forms. Users could then check off the tables they wished to search for and click a

“Submit” button, which would then open up the query page with the appropriate field already filled in. This would save users time, especially with the table names that are hard to remember.

Due to constraints in time, the enhancement has not yet been fully implemented in Perl at the time of this report. However, a JavaScript outline can be seen below – opening the query page with the appropriate field already filled in and allowing multiple entries are the only things left to be completed.

```
<html><head>

<script language=javascript>
<!--
var field = "";
function updateSV(val) {
    field = val;
}
function setField(what) {
    what.myText.value = field;
}
//-->
</script>

</head><form>

<table border="2">
<em>sf0807</em>
<tr>
<td><input type="radio" ...
    onClick=updateSV("artifact")></td>
<td><a href="/cgi-bin/treq.pl?uschema
=sf0807&utable=artifact">artifact</a>
</td></tr>
...
<tr>
<td><input type="radio" ...
    onClick=updateSV("user_role")></td>
<td><a href="/cgi-bin/treq.pl?uschema
=sf0807&utable=user_role">user_role</a>
</td></tr>
<tr>
<td><input type="radio" ...
    onClick=updateSV("users")></td>
<td><a href="/cgi-bin/treq.pl?uschema
=sf0807&utable=users">users</a>
</td></tr>
</table>
</form>

<input onClick="setField(this.form)"
type=button value="Query Field"><br><br>

</center></body></html>
```

sf0807

<input type="radio"/> artifact
<input type="radio"/> artifact canned responses
<input type="radio"/> artifact category
<input type="radio"/> artifact counts agg
<input type="radio"/> artifact file
<input type="radio"/> artifact group
<input type="radio"/> artifact group list
<input type="radio"/> artifact history
<input type="radio"/> artifact message
<input type="radio"/> artifact perm
<input type="radio"/> artifact resolution
<input type="radio"/> artifact status
<input type="radio"/> db images
<input type="radio"/> doc data
...
<input type="radio"/> user group
<input type="radio"/> user perms
<input type="radio"/> user role
<input type="radio"/> users

Query Field

Final Summary

The second half of this research report served as an overview of the University of Notre Dame’s Zerlot open source data repository, and how the data and information presented at Zerlot can be optimized for those interested in tracking the trends and patterns of the open source community that were discussed earlier.

It is admirable to see how open source software has managed to grow as the pace it has over the past few decades. With further development efforts and international expansion, it will continue to vie for position against proprietary vendors.

Sources

- [1] B. Perens, “The Open Source Definition”.
- [2] R. Stallman, “The GNU Operating System and the Free Software Movement”.
- [3] L. Torvalds, “The Linux Edge”.
- [4] C. DiBona, “Open Source and Proprietary Software Development”.
- [5] J. Feller and B. Fitzgerald, “Understanding Open Source Software”.
- [6] M. Baker, “The Mozilla Project: Past and Future”.
- [7] <http://www.cygwin.com>
- [8] J. Allison, “A Tale of Two Standards”.
- [9] D. Joyner, “Mathematical Open Source Software”.
- [10] M. Tiemann, “The Future of Cygnus Solutions – An Entrepreneur’s Account”.
- [11] B. Laurie, “Open Source and Security”.
- [12] S. Weber, “Patterns of Governance in Open Source”.
- [13] B. Behlendorf, “Open Source as a Business Strategy”.
- [14] wikipedia.org/wiki/Information_architecture