To Study the Open Sources and its Applications in Industry

Author Name: - Gurlal Singh
Guide Name: - Mr. Ashwani Sethi
College: - Guru Nanak College Killianwali
Email Id: - Ls.Sidhu@Ymail.Com

ABSTRACT

Open source is software developed by uncoordinated but loosely collaborating programmers, using freely distributed source code and the communications infrastructure of the Internet. Open source has a long history rooted in the Hacker Ethic. The term open source was adopted in large part because of the ambiguous nature of free software. Various categories of free and non-free software are commonly referenced, some with interchangeable meanings. Several licensing agreements have therefore been developed to formalize distribution terms. The Cathedral and the Bazaar is the most frequently cited description of the open-source development methodology, however although the paper identifies many mechanisms of successful open-source development, it does not expose the dynamics. There are literally hundreds, if not thousands, of open-source projects currently in existence.

The term Open Source is widely applied to describe some software development methodologies. This paper does not provide a judgment on the open source approach, but exposes the fact that simply stating that a project is open source does not provide a precise description of the approach used to support the project. By taking a multidisciplinary point of view, we propose a collection of characteristics that are common, as well as some that vary among open source projects. The set of open source characteristics we found can be used as a tick-list both for analyzing and for setting up open source projects. Our tick-list also provides a starting point for understanding the many meanings of the term open source.

Keywords: -

Open Source Software; Software Process; Software Business Models; Information Systems (IS) Development

INTRODUCTION

Open-source software (OSS) is computer software with its source code made available with a license in which the copyright holder provides the rights to study change and distribute the software to anyone and for any purpose. Open-source software is very often developed in a public, collaborative manner. Open-source software is the most prominent example of open-source development and often compared to (technically defined) user-generated content or (legally defined) open-content movements.

A report by the Standish Group (from 2008) states that adoption of open-source software models has resulted in savings of about $60 billion per year to consumers.

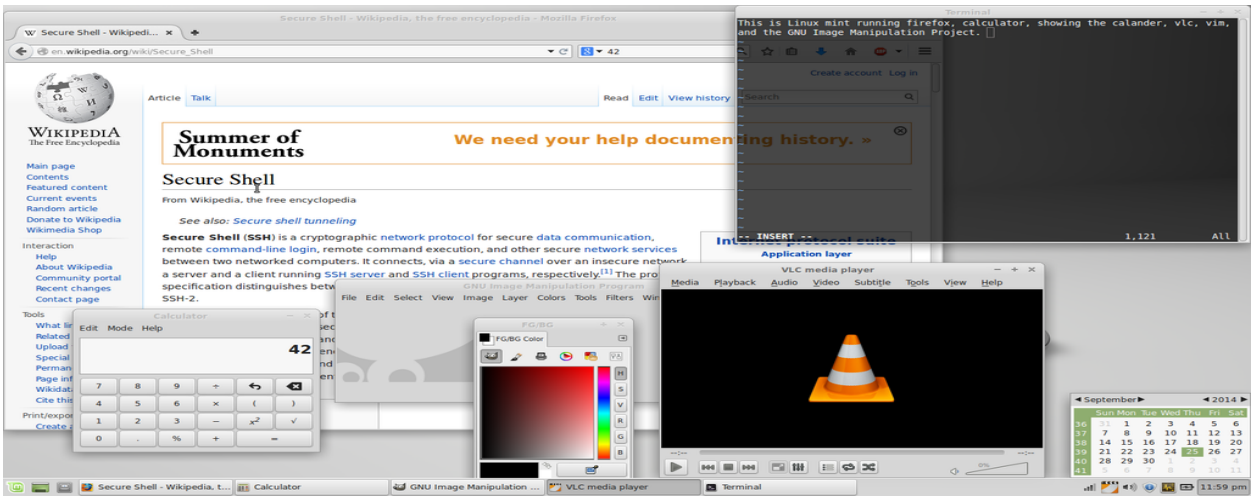

Figure 1.1: A screenshot of Linux Mint running the Xfce desktop environment, Firefox, a calculator program, the builtin calendar, Vim, GIMP, and VLC media player, all of which are open source software.

1.1 DEFINITIONS

The Open Source Initiative's (OSI) definition is recognized as the standard or de facto definition. Eric S. Raymond and Bruce Perens formed the organization in February 1998. With about 20 years of evidence from case histories of closed and open development already provided by the Internet, OSI continued to present the "open source" case to commercial businesses. They sought to bring a higher profile to the practical benefits of freely available source code, and wanted to bring major software businesses and other high-tech industries into open source.

OSI uses The Open Source Definition to determine whether it considers a software license open source. The definition was based on the Debian Free Software Guidelines, written and adapted primarily by Perens. Perens did not base his writing on the "four freedoms" of Free Software from the Free Software Foundation (FSF), which were only widely available later.

Figure 1.2: the logo of the Open Source Initiative

1.1.1 PROLIFERATION OF THE TERM

While the term "open source" applied originally only to the source code of software, it is now being applied to many other areas such as Open source ecology, a movement to decentralize technologies so that any human can use them. However, it is often misapplied to other areas which have different and competing principles, which overlap only partially.

1.1.2 OPEN SOFTWARE LICENSING

A license defines the rights and obligations that a licensor grants to a licensee. Open source licenses grant licensees the right to copy, modify and redistribute source code (or content). These licenses may also impose obligations (e.g., modifications to the code that are distributed must be made available in source code form, an author attribution must be placed in a program/ documentation using that open source).

Authors initially derive a right to grant a license to their work based on the legal theory that upon creation of a work the author owns the copyright in that work. What the author/licensor is granting when they grant a license to copy, modify and redistribute their work is the right to use the author's copyrights. The author still retains ownership of those copyrights; the licensee simply is allowed to use those rights, as granted in the license, so long as they maintain the obligations of the license. The author does have the option to sell/assign, versus license, their exclusive right to the copyrights to their work; whereupon the new owner/assignee controls the copyrights. The ownership of the copyright (the "rights") is separate and distinct from the ownership of the work (the "thing") – a person can own a copy of a piece of code (or a copy of a book) without the rights to copy, modify or redistribute copies of it.

When an author contributes code to an open source project (e.g., Apache.org) they do so under an explicit license (e.g., the Apache Contributor License Agreement) or an implicit license (e.g., the open source license under which the project is already licensing code). Some open source projects do not take contributed code under a license, but actually require (joint) assignment of the author's copyright in order to accept code contributions into the project (e.g., OpenOffice.org and its Joint Copyright Assignment agreement).

Placing code (or content) in the public domain is a way of waiving an author's (or owner's) copyrights in that work. No license is granted, and none is needed, to copy, modify or redistribute a work in the public domain.

Examples of free software license / open source licenses include Apache License, BSD license, GNU General Public License, GNU Lesser General Public License, MIT License, Eclipse Public License and Mozilla Public License.

The proliferation of open-source licenses is one of the few negative aspects of the open-source movement because it is often difficult to understand the legal implications of the differences between licenses. With more than 180,000 open source projects available and its more than 1400 unique licenses, the complexity of deciding how to manage open-source usage within "closed-source" commercial enterprises have dramatically increased. Some are home-grown while others are modeled after mainstream FOSS licenses such as Berkeley Software Distribution ("BSD"), Apache, MIT-style (Massachusetts Institute of Technology), or GNU General Public License ("GPL"). In view of this, open source practitioners are starting to use classification schemes in which FOSS licenses are grouped (typically based on the existence and obligations imposed by the copyleft provision; the strength of the copyleft provision).

An important legal milestone for the open source / free software movement was passed in 2008, when the US federal appeals court ruled that free software licenses definitely do set legally binding conditions on the use of copyrighted work, and they are therefore enforceable under existing copyright law. As a result, if end-users do violate the licensing

conditions, their license disappears, meaning they are infringing copyright.

1.1.3 CERTIFICATIONS

Certification can help to build higher user confidence. Certification could be applied to the simplest component that can be used by developers to build the simplest module to a whole software system. There have been numerous institutions involving in this area of the open source software including The International Institute of Software Technology / United Nations University. UNU/IIST is a non-profit research and education institution of The United Nations. It is currently involved in a project known as "The Global Desktop Project". This project aims to build a desktop interface that every end-user is able to understand and interact with, thus crossing the language and cultural barriers. It is drawing huge attention from parties involved in areas ranging from application development to localization. Furthermore, this project will improve developing nations' access to information systems. UNU/IIST aims to achieve this without any compromise in the quality of the software It believes a global standard can be maintained by introducing certifications and is currently organizing conferences in order to explore frontiers in the field.

Alternatively, assurance models (such as DO178B) have already solved the "certification" approach for software. This approach is tailorable and can be applied to OSS, but only if the requisite planning and execution, design, test and traceability artifacts are generated.

1.2 OPEN-SOURCE SOFTWARE DEVELOPMENT

1.2.1 DEVELOPMENT MODEL

In his 1997 essay The Cathedral and the Bazaar, open-source evangelist Eric S. Raymond suggests a model for developing OSS known as the bazaar model. Raymond likens the development of software by traditional methodologies to building a cathedral, "carefully crafted by individual wizards or small bands of mages working in splendid isolation". He suggests that all software should be developed using the bazaar style, which he described as "a great babbling bazaar of differing agendas and approaches."

In the traditional model of development, which he called the cathedral model; development takes place in a centralized way. Roles are clearly defined. Roles include people dedicated to designing (the architects), people responsible for managing the project, and people responsible for implementation. Traditional software engineering follows the cathedral model. Fred P. Brooks in his book The Mythical Man-Month advocates this model. He goes further to say that in order to preserve the architectural integrity of a system; the system design should be done by as few architects as possible.

The bazaar model, however, is different. In this model, roles are not clearly defined. Gregorio Robles suggests that software developed using the bazaar model should exhibit the following patterns:

Users should be treated as co-developers

The users are treated like co-developers and so they should have access to the source code of the software. Furthermore users are encouraged to submit additions to the software, code fixes for the software, bug reports, documentation etc. Having more co-developers increases the rate at which the software evolves. Lanus's law states, "Given enough eyeballs all bugs are shallow." This means that if many users view the source

code, they will eventually find all bugs and suggest how to fix them. Note that some users have advanced programming skills, and furthermore, each user's machine provides an additional testing environment. This new testing environment offers that ability to find and fix a new bug.

Early releases

The first version of the software should be released as early as possible so as to increase one's chances of finding co-developers early.

Frequent integration

Code changes should be integrated (merged into a shared code base) as often as possible so as to avoid the overhead of fixing a large number of bugs at the end of the project life cycle. Some open source projects have nightly builds where integration is done automatically on a daily basis.

Several versions

There should be at least two versions of the software. There should be a buggier version with more features and a more stable version with fewer features. The buggy version (also called the development version) is for users who want the immediate use of the latest features, and are willing to accept the risk of using code that is not yet thoroughly tested. The users can then act as co-developers, reporting bugs and providing bug fixes.

High modularization

The general structure of the software should be modular allowing for parallel development on independent components.

Dynamic decision making structure

There is a need for a decision making structure, whether formal or informal, that makes strategic decisions depending on changing user requirements and other factors. Cf. Extreme programming.

Data suggests, however, that OSS is not quite as democratic as the bazaar model suggests. An analysis of five billion bytes of free/open source code by 31,999 developers shows that 74% of the code was written by the most active 10% of authors. The average number of authors involved in a project was 5.1, with the median at 2.

PROBLEM FORMULATION

- Before developing research we keep following things in mind so that we can develop powerful and quality research.

3.1 PROBLEM STATEMENT

- Open-source software can be sold and used in general commercially. Also, commercial open-source applications are a part of the software industry for some time. Despite that, except for Red Hat and VA Software, no other pure open-source company has gone public on the major stock markets. While commercialization or funding of open-source software projects is possible, it is considered challenging.
- Since several open-source licenses stipulate that derived works must distribute their intellectual property under an open-source (copyleft) license, ISVs and VARs have to develop new legal and technical mechanisms to foster their commercial goals, as many traditional mechanisms are not directly applicable anymore.
- Traditional business wisdom suggests that a company's methods, assets, and intellectual properties

should remain concealed from market competitors as long as possible to maximize the profitable commercialization time of a new product. [According to whom?] Open-source software development minimizes the effectiveness of this tactic; development of the product is usually performed in view of the public, allowing competing projects or clones to incorporate new features or improvements as soon as the public code repository is updated, as permitted by most open-source licenses. Also in the computer hardware domain, a hardware producer who provides free and open software drivers reveals the knowledge about hardware implementation details to competitors, who might use this knowledge to catch up.

- Therefore, there is considerable debate about whether vendors can make a sustainable business from an open-source strategy. In terms of a traditional software company, this is probably the wrong question to ask. Looking at the landscape of open source applications, many of the larger ones are sponsored (and largely written) by system companies such as IBM who may not have an objective of software license revenues. Other software companies, such as Oracle and Google, have sponsored or delivered significant open-source code bases. These firms' motivation tends to be more strategic, in the sense that they are trying to change the rules of a marketplace and reduce the influence of vendors such as Microsoft. Smaller vendors doing open-source work may be less concerned with immediate revenue growth than developing a large and loyal community, which may be the basis of a corporate valuation at merger time.
- A variety of open-source compatible business approaches have gained prominence in recent years [according to whom?]; notable examples include dual licensing, software as a service, not charging for the software but for services, fermium, donation-based funding, and crowd funding.
- The underlying objective of these business models is to harness the size and international scope of the open-source community (typically more than an order of magnitude larger than what would be achieved with closed-source models) for a sustainable commercial venture.[citation needed] The vast majority of commercial open-source companies experience a conversion ratio (as measured by the percentage of downloader's who buy something) well below 1%, so low-cost and highly-scalable marketing and sales functions are key to these firms' profitability.

OBJECTIVE

Software development requires much knowledge and work. I wonder why useful software such as Mozilla and VideoLAN are made free for download. Much free software tends to be very good indeed. I'm not against free software, though. I also benefit from them. Free software is developed and given away normally with an option to donate to help with development costs.

Open source software is developed by groups of people that contribute different features and functions to an application or operating system.

Take Linux for example. There are many versions of Linux that have been contributed too over the years, but the underlying code is very similar and uses a Linux Kernel as the basis for the OS.

Free software model in the context of your question liberates the revenue model from the software product. You are no more just charging for a product, although you can still charge for the product. For example if shoe-making was open sourced. You'd not just sell shoes, but also the design blueprint for it. How do you gain the upper hand? If you're the person with original plan, everyone down the line credits you. Buyers know who the original person who knows the stuff is. If you're one who bought the shoes and now designed your derivative, they'd sell based on whats the speciality of your derivative. You'd realize that setting up shop would require capital and it's somewhat true for open source software. All major successful free software has the biggest corporations FUNDING the labor towards developing them. VideoLAN doesn't exactly enjoy a prominent corp backing, so their development on Mac had/still has come down to a crawl.

I would add an example of open source ERP. OpenERP is a comprehensive suite of business applications and has a modular approach which allows customers to start with one application and then adds other modules as they go. It's license free, customizable and very easy to use. The product has gained a lot of popularity due to its no license policy and the verity of solutions it offers. To which its community can contribute to develop and improve. To know more about the line of solutions OpenERP offers, follow this link: http://bit.ly/aUeAZu.

The main objective of this research is to study the open sources and its applications used in the industry.

RESEARCH METHODOLOGY

4.1 METHODOLOGY

The Cathedral and the Bazaar is the most frequently cited description of the open-source development methodology. Eric Raymond's discussion of the Linux development MODEL as applied to a small project is a useful commentary. However, it should be noted that although the paper identifies many mechanisms of successful open-source development, it does not expose the dynamics. In this sense, the description is inherently weak.

4.1.1 Plausible Promise

Raymond remarks that it would be difficult to originate a project in bazaar mode. To build a community, a program must first demonstrate plausible promise. The implementation can be crude or incomplete, but it must convince others of its potential. This is given as a necessary precondition of the bazaar, or open-source, style.

Interestingly, many COMMERCIAL SOFTWARE companies use this approach to ship software products. Microsoft, for example, consistently ships early versions of products that are notoriously bug ridden. However as long as a product can demonstrate plausible promise, either by setting a standard or uniquely satisfying a potential need, it is not necessary for early versions to be particularly strong.

Critics suggest that the effective utilization of bazaar principles by closed source developers implies ambiguity. Specifically that the Cathedral and the Bazaar does not sufficiently describe certain aspects of the open-source development process.

4.1.2 Release Early, Release Often

Early and frequent releases are critical to open-source development. Improvements in functionality are incremental, allowing for rapid evolution, and developers are "rewarded by the sight of constant improvement in their work."

Product evolution and incremental development are not new. Mills initially proposed that any software system should be grown by incremental development (Mills, 1971). Brooks would later elaborate on this concept, suggesting that developers should grow rather than build software, adding more functions to systems as they are run, used, and tested (Brooks, 1986). Basili suggested the concept of iterative enhancement in large-scale software development (Basili and Turner, 1975), and Boehm proposed the spiral MODEL, an evolutionary prototyping approach incorporating risk management.

Let's have a look at the general diagram in a different way to see what is running concurrently: Release Early, Release Often

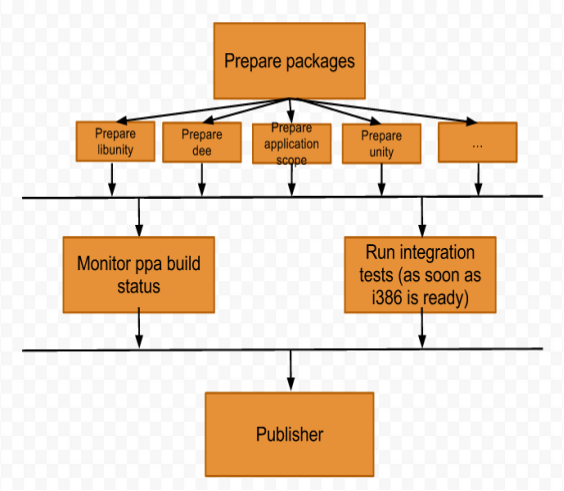

Figure 4.1: General Diagram In A Different Way To See What Is Running Concurrently: Release Early, Release Often

Open source relies on the Internet to noticeably shorten the iterative cycle. Raymond notes that "it wasn't unknown for [Linux] to release a new kernel more than once a day." (Raymond, 1998a) Mechanisms for efficient distribution and rapid feedback make this practice effective.

However, successful application of an evolutionary approach is highly dependent on a modular architecture. Weak modularity compromises change impact and minimizes the effectiveness of individual contributors. In this respect, projects that do not encourage a modular architecture may not be suitable for open-source development. This contradicts Raymond's underlying assertion, that open source is a universally better approach.

4.1.3 Debugging is Parallelizable

Raymond emphasizes large-scale peer review as the fundamental difference underlying the cathedral and bazaar styles. The bazaar style assumes that "given a large enough beta-tester and co-developer base, almost every problem will be characterized quickly and the fix obvious to someone." Debugging requires less coordination relative to development, and thus is not subject "to the same quadratic complexity and management costs that make adding developers problematic." (Raymond, 1998a)

The basic premise is that more debuggers will contribute to a shorter test cycle without significant additional cost. In other words, "more users find more bugs because adding more users adds more ways of stressing the program." (Raymond, 1998a) However, open source is not a prerequisite for peer review. For instance, various forms of peer review are commonly employed in SOFTWARE ENGINEERING. The question might then become one of scale, but Microsoft practices beta-testing on a

scale matched only by larger open-source projects.

Raymond continues, suggesting that debugging is even more efficient when users are co-developers, as is most often the case in open-source projects. This is also subject to debate. Raymond notes that each tester "approaches the task of bug characterization with a slightly different perceptual set and analytical toolkit, a different angle on the problem." (Raymond, 1998a) This is characterized by the fact that developers and end-users evaluate products in very different ways. It therefore seems likely that peer review under the bazaar MODEL would be constrained by a disproportionate number of co-developers.

EXPERIMENTAL RESULTS

5.4 THE GROWTH OF OPEN SOURCE

Open source software is having a major impact on the software industry and its production processes. Many software products today contain at least some open source software components. Some commercial products are completely open source software. In some markets, for example, web servers, open source software hold a dominant market share.

Open source software today has a strong presence in industry and government. Walli et al. observe: "Organizations are saving millions of dollars on IT by using open source software. In 2004, open source software saved large companies (with annual revenue of over $1 billion) an average of $3.3 million. Medium-sized companies (between $50 million and $1 billion in annual revenue) saved an average $1.1 million. Firms with revenues under $50 million saved an average $520,000."

Commercially, the significance and growth of open source is measured in terms of revenue generated from it. Lawton and Notarfonzo state that packaged open source applications generated revenues of $1.8 billion in 2006. The software division of the Software & Information Industry Association estimates that total packaged software revenues were $235 billion in 2006. Thus, open source revenue, while still small compared to the overall market (~0.7%) is not trivial any longer.

However, open source software today is part of many proprietary (closed) source products, and measuring its growth solely by packaged software revenue is likely to underestimate its size and growth by a wide margin. To measure the growth of open source we need to look at the total growth of open source projects and their source code.

Several studies have been undertaken to measure the growth and evolution of individual open source software projects. Most of these studies are exemplary, focusing on a few selected projects only. The exception is Koch's work, which uses a large sample (>4000 projects) to determine overall growth patterns in open source projects, concluding that polynomial growth patterns provide good models for these projects. Such work is mostly motivated by trying to understand how individual open source projects grow and evolve.

The work presented in this paper, in contrast, analyzes the overall growth of open source, aggregating data from more than 5000 active and popular open source projects to determine the total growth of source code and number of projects. Assuming a positive correlation between work spent on open source, its total growth in terms of code and number of projects, and the revenue generated from it, understanding the overall growth of open source will give

us a better indication of how significant a role open source will play in the future.

Understanding overall open source growth helps more easily answer questions about, for example, future product structures (how much code of an application is likely to be open source code?), labor economics (how much and which open source skills does a company need?), and revenue (what percentage of the software market's revenue will come from open source?).

The work presented in this paper shows that the total amount of open source code and the total number of projects is growing exponentially. Assuming a base of 0.7% of the market's revenue, exponential growth is a strong indicator that open source will be of significantly increasing commercial importance. The remainder of this paper discusses our study and validates the hypothesis of exponential growth of open source.

However, we cannot unambiguously identify situations where a developer adds redundant source code to the code base. Copy and paste is a common practice in software development, independently of whether it is internal, external, planned or opportunistic. To deal with this issue, we adopt two approaches.

1. In the first approach we ignore the copy and paste problem and analyze the source lines of code added. The argument is that copy and paste is a reality of software development and that the copied code is part of the project. Hence, copy and paste simply needs to be accepted.
2. In the second approach we find the average and the standard deviation for the code added over time. We ignore all commits where lines of code added is greater than average code added per commit plus three times the standard deviation. The heuristic's assumption is that by not considering such large commits we ignore all commits based on copy and paste.

An analysis of average code contribution size in commits provides a cut-off value of 3060 SLoC that we use for the heuristic. This second approach is conservative in that we ignore not only copy and paste but also commits containing new code added. So we err on the lower side of total open source contributions.

We employ these two approaches to get an upper and a lower bound for the growth in source lines of code and number of projects. We can therefore say that properties like the exponential growth observed in both the upper and lower bound curve apply to the real curve as well.

5.5 ANALYSIS AND RESULTS

We first analyze growth rate and total growth in open source software code and then analyze growth rate and total growth in open source software projects.

5.5.1 Growth in source code

Figures 1 and 2 show plots that represent the growth in source lines of code added using Approach 1 and 2 respectively. The Y-axis shows the number of lines of code added each month and the X-axis shows the time. Each data point on the plot represents the total number of lines of code added during that month. The time frame is 1995 through 2006 for all projects. We can see an upward trend in the amount of code added over time. Both Approach 1 and 2 show a similar pattern of growth.

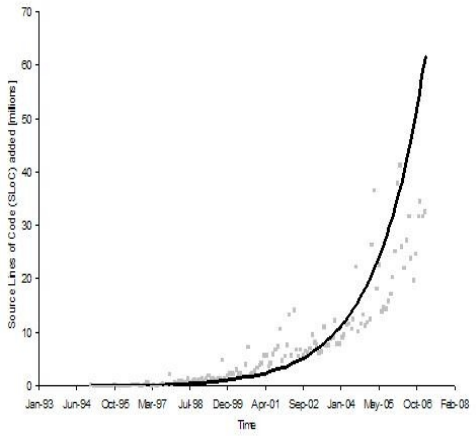

Figure5.1: Graph of source lines of code added [millions] (Approach 1)

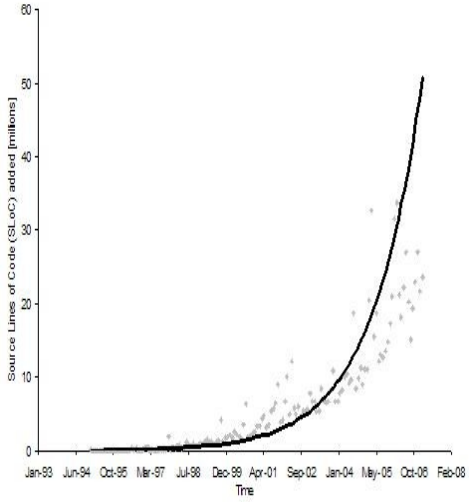

Figure 5.2: Graph of source lines of code added [millions] (Approach 2)

Table shows models for the two plots. In both cases, the best fitting model is an exponential curve with an R-square value of about 0.9, giving us confidence in the validity of the claim that the amount of code added is growing exponentially.

Table 5.1: Model of source lines of code added

Approach	Model	R-square value
1	$y = 70833 * e^{0.0464x}$	0.901
2	$y = 64004 * e^{0.046x}$	0.897

where,

y: Source lines of open source code added

x: Time from Jan 1995 to Dec 2006 in months

Figure 3 shows the total number of lines of open source code over time. Table 2 shows the statistical models for the two approaches. The doubling time for Approach 1 is 12.5 months, and the doubling time for Approach 2 is 14.9 months. We observe that the total code in Approach 2 is lower than in Approach 1 but follows a similar trend. This behavior is expected as we eliminated all large commits in the second approach to exclude copy and paste contributions.

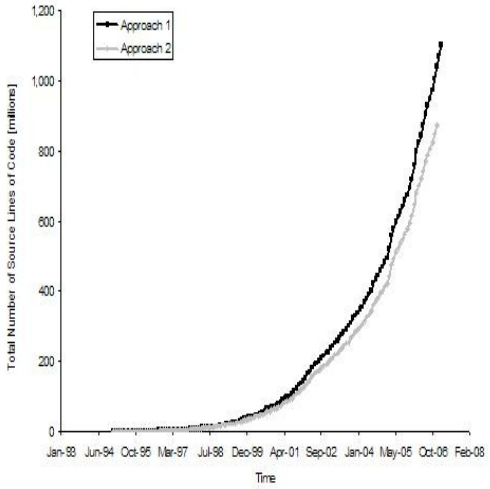

Figure 5.3: Graph of total source lines of code [millions] (both approaches)

Table 5.2: Model of total source lines of code

Approach	Model	R-square value
1	$y = 784098 * e^{0.0555x}$	0.961
2	$y = 2E+06 * e^{0.0464x}$	0.964
where, y: Total open source lines of code x: Time from Jan 1995 to Dec 2006 in months		

5.5.2 GROWTH IN OPEN SOURCE

Figure 4 shows the number of projects added over time and Table 3 shows the model and its fit with the data. For each project, there is a first occurrence of a project action (for example, the initial commit action), and that point of time is considered the birth date of the project. This is the point of time when the project is counted as added to the overall set of projects.

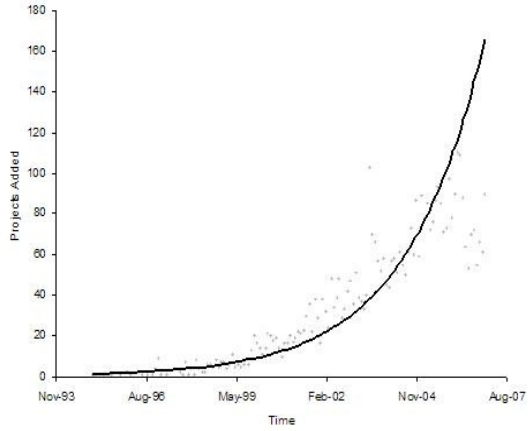

Figure 5.4: Graph of number of open source projects added

Table 5.3: Model of number of open source projects added

Model	R-square value
$y = 1.0641e^{0.035x}$	0.884
where, y: Total number of open source projects x: Time from Jan 1995 to Dec 2006 in months	

Large distributions like Debian are counted as one project. Popular projects such as GNU Emacs are counted as projects of their own, little known or obsolete packages such as the Zoo archive utility are ignored. Many of the projects that were included in a Debian distribution around 1998 are not popular enough today (as stand-alone projects) to be included in our copy of the Ohloh database. And again, we get the best fit for the resulting curve for an exponential model with an R-square value of 0.88. Figure 5 then shows the total number of projects and Table 4 shows the corresponding model and its fit with the data. Again, we get the best fit for an exponential model with an R-square value of 0.96. The doubling time is 13.9 months.

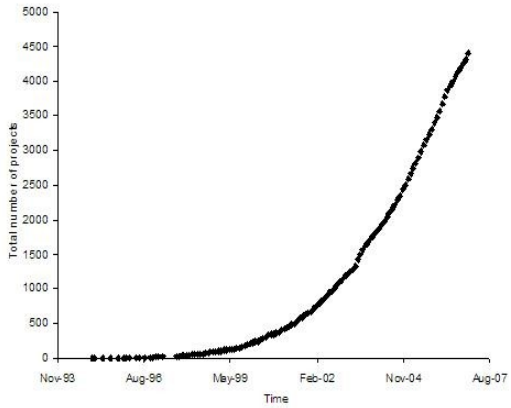

Figure 5.5: Graph of total number of open source projects

Table 5.4: Model of total number of open source projects

Model	R-square value
$y = 7.1511e^{0.0499x}$	0.956
where, y: Total number of open source projects x: Time from Jan 1995 to Dec 2006 in months	

5.5.3 REVIEW OF FINDINGS

This section shows the growth of source code in open source projects as well as the growth of open source projects itself. We consistently get the best fit for the data using exponential models. The doubling time based on the exponential models is about 14 months for both the total amount of source code and the total number of projects. It should be noted that if we were to break up the data sets into separate time periods, we might find better fits for other models than the exponential model. In future work we will analyze the overall growth in distinct phases, each of which is best explained by a separate growth model.

We discuss the size and frequency of code contributions to open source projects. We can use those results to further increase our confidence in the results presented above. Specifically, the lines of code added can be assumed equal to the product of the average size of a commit in terms of source lines of code and the commit frequency. Our analysis shows that the average commit size is almost constant while the commit frequency (number of commits per week) increases exponentially between Jan 1995 to Dec 2006. This verifies our findings about the exponential growth in open source.

CONCLUSION AND FUTURE WORK

This chapter is based upon the conclusion of what we have done so far and how the system can be further enhanced with an increase in requirements.

6.1 CONCLUSION

Open source is software developed by uncoordinated but loosely collaborating programmers, using freely distributed source code and the communications infrastructure of the Internet. Open source is based on the philosophy of free software. However, open source extends this ideology slightly to present a more commercial approach that includes both a business model and development methodology. Various categories of free and non-free software are commonly, and incorrectly, referenced, including public domain, freeware, and shareware. Licensing agreements such as the GPL have been developed to formalize distribution terms. The Open Source Definition provides a framework for evaluating these licenses.

There are hundreds, if not thousands, of open-source projects currently in existence. These projects face growing challenges in terms of scalability and inherently weak tool support. However open source is a pragmatic example of software development over the Internet

The significance of open source has been continuously increasing over time. Our research validates this claim by looking at the total growth of open source. Our work shows that the additions to open source projects, the total project size (measured in source lines of code), the number of new open source projects, and the total number of open source projects are growing at an exponential rate. The total amount of source

code and the total number of projects double about every 14 months.

Our results open gates for further research around the growth of open source and the acceptance of open source in industry and government. Future research should explore questions like what factors are influencing this exponential growth, how source code growth relates to the number of engaged software developers, and whether or how long open source can sustain this exponential growth.

REFERENCES

[1]. Mockus, A. AT&T Bell Labs. Naperville, IL, USA "A case study of open source software development: the Apache server" Date of Conference: 4-11 June 2000, Pages 263-272, Print ISBN: 1-58113-206-9, INSPEC Accession Number: 6734866

[2]. Joseph Feller, Brian Fitzgerald. "A Framework Analysis of the Open Source Software Development Paradigm" Published In ICIS '00 Proceedings of the twenty first international conferences on Information System, Pages 58 - 69 Publication Date: 10 Dec 2012

[3]. Stefan Koch and Georg Schneider, "Effort, Co-Operation and Co-Ordination in an Open Source Software Project: GNOME" Date of Conference: 8 Feb. 2002, Pages: 27 – 42

[4]. Yunwen Ye, "Toward an understanding of the motivation of open source software developers" Date of Conference: 3-10 May 2003, Pages 419 - 429, Print ISBN: 0-7695-1877-X, INSPEC Accession Number: 8064388).

[5]. Godfrey, M.W; "Evolution in open source software: a case study" Publication Date: 11-14 Oct 2000, Pages 131 - 142, E-ISBN: 1063-6773, Print ISBN: 0-7695-0753-0, INSPEC Accession Number: 6771737)

[6]. Georg von Krogh. ; "Community, joining, and specialization in open source software innovation: a case study" Date of Conference: 19 June 2003, Pages 1141 – 1152

[7]. Guido Hertel, Sven Niedner; "Motivation of software developers in Open Source projects" Date of Conference: 12 April 2003, Pages 141 – 152

[8] Georg Von Krogh; "Special issue on open source software development" Vol 32, Issue 7, Date of Conference: July 2003, Pages 1149 – 1157

[9] Andrea Bonaccorsi, Cristina Rossi; "Why Open Source software can succeed" Vol 32, Issue 7, and Date of Conference: July 2003, Pages 1243 – 1258

[10] Siobhán O'Mahony; "Guarding the commons: how community managed software projects protect their work" Vol 32, Issue 7, Date of Conference: 28 May 2003, Pages 1179 – 1198

[11] David Zeitlyn; "Gift economies in the development of open source software: anthropological reflections" Vol 32, Issue 7, Date of Conference: 10 April 2003, Pages 1287 – 1291

[12] Nikolaus Franke, Eric von Hippel; "Satisfying heterogeneous user needs via innovation toolkits: the case of Apache security software" Vol 32, Issue 7, Date of Conference: 29 May 2003, Pages 1199 – 1215

[13] Audris Mockus; "A case study of open source software development: the Apache server" ICSE '00 Proceedings of the 22nd international conference on Software, Date of Conference: 1 June 2000, Pages 263 – 272

[14] Gacek, C; "The many meanings of open source" Publication Date: 9 Aug 2004, Pages 34 - 40, Print ISBN: 0740-7459, INSPEC Accession Number: 7949988)