CS 5212 Software Project Management

Project Report – Research Paper

STUDY ON MANAGEMENT OF OPEN SOURCE SOFTWARE PROJECTS

By

Krishnan Nair Srijith (HT006458L)

(srijith@comp.nus.edu.sg)

October 2002

ABSTRACT

The ideology of open source software development is spearheading a shift in the way we approach the process of software development. Not only is it changing the perception of costs associated with projects, but also the management aspect of these development processes. The management of open source projects is very different from a traditional project due to the inherent nature of the objectives, team structure and the benefits involved. Several studies have examined various issues related to the management of these proje cts. However there is a lack of a study that puts together all the findings so that the interrelationships between these findings can be explored. This study tries to overcome this shortcoming and present the findings of other studies in a comprehensive ma nner and at the same time look at the entire process from a bird' eye point of view.

TABLE OF CONTENT

"Their community is very, very good, and we're hard at work trying to follow that model"
Jim Allchin, Group Vice President, Microsoft

1. Introduction

Even though open source software (OSS) projects have been producing quality, reliable and stable products for several years now, only recently have interest grown in the academic circles to study the process, due to the success of several OSS products that have received wide spread usage and have become market leaders in their fields.

The Open Source Initiative (OSI) [1] proposes a definition for "Open Source Software" as, among other things, software that, "include(s) source code, and must allow distribution in source code as well as compiled form", does not "restrict any party from selling or giving away the software as a component of an aggregate software distribution containing programs from several different sources. The license shall not require a royalty or other fee for such sale." At the time of writing of this report, 35 different licenses have been OSI certified as open source. There seems to be some confusion between the phrases "free software" and "open source software". Free software is a more restricted subset of OSS and is one of the 35 licenses approved by OSI. In this study, any project that uses a license approved by OSI is considered as an OSS project.

Several studies have been conducted to investigate the characteristic and nature of development of software using the OSS paradigm. These studies have reported various interesting and sometimes contradicting results. This report aims to survey some of these studies and report their finding in a comprehensive manner.

Section 2 examines the "Bazaar" model of open source projects and its implications. The requirements management process of OSS projects is studied in section 3 and configuration management in section 4. The effect of OSS projects phenomenon on Brooks' Law is studied next in section 5. Leadership plays an important role in the project management and is looked into Section 6. Section 7 summarises the discussions in the paper and concludes with the major findings.

"…Linux community seemed to resemble a great babbling bazaar of differing agendas and approaches…"

Eric S. Raymond, [2]

2. The Bazaar model

One of the most well know and earliest work that tried to study and classify the working of OSS projects was "The Cathedral and the Bazaar" by Eric S. Raymond. In 1999, this work along with other writings on Eric Raymond was published as a book "The Cathedral & the Bazaar" [2]. In this momentous paper, Raymond compares the working of a closed source, proprietary project to that of a highly organised Cathedral and OSS projects to that of a chaotic Bazaar. Based on the example of one of the most successful OSS project Linux, and his own experience with the development of the mail utility *Fetchmail* [3], Raymond proposes some interesting and some times controversial ideas, which he calls "lessons". Some of these "lessons" that have direct relationship with software project management concepts or theory are listed below:

- "Every good work of software starts by scratching a developer's personal itch."

  This "lesson" is a characteristic of most OSS projects. Most OSS project was started by individuals or groups either because a product that met their functional specifications were not available or were too costly to be used. This is similar to the conclusion reached by U. Asklund and L. Bendix in [4]. They say, "OSS projects are run by developers for developers and there is no interest or resources for bureaucratic overhead".

- "Release early. Release often. And listen to your customers."

  This is one of the most important differences between the Cathedral model of software development and the Bazaar style. In the Cathedral style, developers work in isolation and release their codes only when it is "totally finished". However the Bazaar model differs in releasing frequent revision to their codes. This process does two things at the same time. It helps in weeding out unstable software fast, thus making the user happy, and at the same time keeps the developers motivated.

- “Given enough eyeballs, all bugs are shallow.”

  Raymond dubs this as “Linus’ Law”, after the primary author of Linux operating system (OS) codes base. It basically means that as long as you have enough people looking for bugs, almost all of them can be caught. This is another difference between the two models. Since the products developed using the Cathedral model is closed, not many are in a position to look for the bugs. The bugs in these projects are thus harder to find and these coupled with the long wait between releases creates dissatisfied users.

  However Nikolai Bezroukov of Fairleigh Dickinson University seems to disagree with this “lesson”. He states in [5] that whatever be the number of developers looking at the codes, bugs that are architectural in nature cannot be debugged. To add on to that, my personal experience shows that even though the number of people who might read through the codes is higher than that for a closed product, most of these people are not skilled enough to spot and debug the bugs that might exist.

- Brooks’ Law does not apply to Internet-based distributed development environment.

  Raymond questions the applicability of Brook’s Law for Linux and other Bazaar based projects. He states “Brooks's Law has been widely regarded as a truism. But we've examined in this essay many number of ways in which the process of open-source development falsifies the assumptions behind it—and, empirically, if Brooks' Law were the whole picture Linux would be impossible.” Raymond puts forward the possibility that using the power of “gift-based”, “ego less programming” and cheap communication, the effect of Brooks’ law can be diminished. He argues that if Brooks’ law was true in all cases, including Linux, the hundreds of volunteers who help in the development of Linux would have reduced the pace of development and increased costs, both of which have not occurred. The appropriateness of Brooks’ Law in OSS project will be discussed in more detail in Section 5.

- Good programmers know what to write. Great ones know what to rewrite (and reuse).

  The code sharing ideology of OSS projects makes it easy to reuse codes and thus save enormous time and effort, which would otherwise be spent in reinventing the wheel.

“Open software development does not adhere to the traditional engineering rationality found in the legacy of software engineering life-cycle models or prescriptive standards.”

W. Scacchi, [6]

3. Requirements management

The requirements management process of OSS projects is different from that of normal closed source projects. W. Scacchi studied the process of requirements management of four OSS projects in [6] and this section of the report details the findings of the paper. It was identified that there are no strict and formal ways of requirements management in OSS projects. Rather, they are ‘implied activities’ that are performed, as a part of discussion on what software should do. The study analyses the various requirements management process as follows:

- Requirements elicitation

The study found out that requirement elicitation is done in a very informal manner, either through emails or bulletin board postings. Most of the time, these requirements are asserted without any specific reference to other supporting documents. One of the reasons why an informal method of requirements elicitation can work is because of the fact that there is a large overlap between the developer population and user populations. It was also reported that no documentation of requirements elicitation process was found in the projects analysed.

- Requirement analysis

It was found that the requirement analysis did not involve any automated analysis or formal reasoning as required by Nuseibeh and Easterbrook in [7]. Rather, they are done via technical narratives and using prior knowledge gathered during the work on similar projects.

- Requirements specification

It was found that requirements for OSS project were specified in medium that referred to or linked to email and bulletin boards. These preliminary specifications are then scrutinised by the developers and then condensed into functional and non-functional specification for the system. An interesting characteristic of OSS projects is that there is usually no recorded formal specification and they are not described using any notational schema or symbols.

- Requirement validation

Since the requirements are never specified formally in a document format or using symbolic methods, there is no formal way to validate these requirements. Rather, the study found out that the informal process of requirements validation was more of a by product of how OSS requirements are described, discussed and linked to other informal system descriptions and implementations.

- Requirements communications

Even though, in all the previous processes of requirements management, the open source projects seem to be less formal than their closed source commercial ones, the process of communication of the requirements to the various members of the team is more well developed in open source projects. It was seen that using the online medium of emails, websites, bulletin boards and shared source code directories, the members could be easily updated on the requirements of the project. This open nature of the specifications allows even outsiders to trace the development and evolution of the project requirements.

In short the study states that there does not exist any formal method of requirements management associated with open source projects. Because of this, the study designates this process as “software informalism”. Rather than rely on hard formalism to denote the requirements of a project, open source projects rely on a web of narratives to formalise, analyse, validate and communicate their requirements to fellow project members and the rest of the society. Even though the requirements management process is informal, it is more open than traditional approach and seems to still fulfil the requirements of the process.

“Change management is probably the weakest part of OSS process”

U. Asklund, L. Bendix, [4]

4. Configuration Management

Most open source projects are run and managed by developers themselves and hence studies on configuration management have been conducted from the point of view of the developers. Asklund and Bendix studied the configuration management process in open source projects in [4]. Instead of using the frame work of configuration identification, control, configuration status accounting and audit, which is generally used when studying typical CS project, they looked at configuration management from the point of view of the developers and used a different framework to analyse the process. They used a framework that looks from the angle of tools used in the configuration of project so that they could obtain a “developer bias” as given below:

- Version Control

Most of the developers of open source projects seem to favour the usage of the tool Concurrent Version System (CVS) [8] as a version control system. All versions and branches are kept at a central repository and developers who want to work on it have to “check out” the codes from this repository and work on it in their own local workspace. The work can be done locally and the developer need not be online during this period. When the coder has finished his coding, he checks the code back into the CVS system. Write access to these repositories are granted sparingly in some project and generously in others.

On the status of version control on the Linux project, the study states “Linux, however, use no tool at all for version control. They simply put the code of each version of the system in a separate directory and apply contributions and patches to a ‘latest’ directory” [4]. However, literature available on various interviews and email copies suggests that this is not the case. The confusion arises because of the fact that the general public does not have any read or write access to the version system used by most of the moderators of the Linux project. According to various emails [9] sent by Linus, it is clear that Linus and a lot of other kernel moderators use the product BitKeeper [10].

- Build Management

Since the usage of the version management tool creates local workspaces, build management is easier as complex issues like linking files and objects from different physical locations are avoided. Tools like 'make' [11] can be used to build the codes. Configuration files to use these tools are also part of the files that can be downloaded onto the local workspace.

- Configuration Selection

In most of the OSS projects studied by the paper, configuration selection was not an issue because of the fact that only the latest releases are maintained. Even where one stable and one developmental releases are being maintained, as in the case of the Linux project, they are, for all practical purposes run as two different processes and there is not much file and configuration sharing between the two projects.

- Workspace management

The CVS tool used enables easy workspace management for the developers. It creates workspaces locally when codes are checked out and the check in of the modified codes is similarly a seamless operation. However when a developer does not have direct write permission to the code repository, he has to submit his changes as patches to the moderators and they check in the code after review.

- Concurrency control

CVS is not primarily meant for concurrency management. It does not have the inbuilt ability to lock files that are checked out. So what it does is that if two users checks out the same piece of code, and after the first guy has made changes and submitted them back, the second coder will be forced to check out the modified version of the code and then integrate his changes into this modified code. When the system cannot do the merge automatically, manual intervention is required. When a conflict does arise, it is usually solved using other communication mediums.

‣ Change Management

In a normal CS project, the process of change management goes as follows. The change is proposed and then it is documented formally and reviewed to decide on its efficacy and technical merit. If it is approved, the change document is passed on to the developers who then implements them and then beta testers test the changes to make sure that they works well.
However, open source projects differ fundamentally from CS project in the above-mentioned process. Since the developers or any other person who wants a functionality into the software can propose changes, instead of waiting for the approval, the changes are usually proposed, documented (in email or bulletin board postings) and then implemented before it is taken up for review. The moderators or the co-ordinators does the evaluation and if approved, it is verified and checked into the proprietary.

This can create long lasting un-updated and unfulfilled "wish lists". Developers choose which changes or whish list features to implement. There is usually no control or responsibility distribution process. As the authors mention "OSS projects would probably benefit from an updated wish list and a better traceability between change request (wish) and the actual changes made to the code". Another problem that can be identified is that the moderators have a large responsibly of protecting the code base form bad patches. However, at the same time, if he is slow in verifying and approving (or discarding) the code changes, he may become the bottleneck and developers may get disillusioned.

On the other hand, even though traditional CS projects emphasise on classifying and prioritising changes requests, there is no inbuilt way in which the changes implemented can be checked for bad implementation issues. As long as the change feature works, the code is approved and no further verification is done. Traditional CS projects can adopt the practise of the OSS community and try to insert an extra step of proper review process after the changes have been implemented.

In short, it can be conjured that the change management of OSS projects are informal and initiated mostly by developers who have some personal interest in the particular changes. Due to the use of modular architecture and tools like CVS, workspace management and distributed work is easily managed. However, the change management process does put the moderators under pressure and heavy responsibility. At the same time, because OSS developers do not support a lot of branches, configuration management becomes easier.

“I don’t know how you take a rule about ‘making a project late’ and apply it to development which has little formal scheduling”

Erik Troan, author of Red Hat RPM

5. Effect on Brooks’ Law

Fredrick Brooks, who headed the team at IBM that created the first large scale computer operating system in 1960s, in his book “The Mythical Man Month” [12] states that, put simply “Adding manpower to a late software makes it later.” This is known as Brooks’ Law. As mentioned in Section 2, Eric Raymond in [2] declared that Brook’s Law was obsolete, if not simply limited. However, in the later editions, he did tone down the implication a bit by stating “I don’t consider Brooks’ Law ‘obsolete’ any more that Newtonian physics is obsolete, just incomplete. Just as you get non-Newtonian effects at high energies and velocities you get non-Brooksian effect when transaction costs go low enough.” [2].

Paul Jones of MetaLab tries to examine this issue in greater details in the work “Brooks’ Law and open source: The more the merrier?” [13]. An interesting twist to this issue is the comment by Erik Troan, the developer of the popular Linux distributor Red Hat’s package manager. He says “I don’t know how you can take a rule about ‘making a late project later’ and apply it to the development which has little formal scheduling”. This basically points to the fact that there are no hard deadlines to be followed by open source projects. Richard Stallman, President of Free Software Foundation [14] agrees. “GNU projects rarely have a schedule”.

Jamie Zawinski, a developer of the Netscape, XEmacs and Mozilla, provides another interesting perspective in the same paper. He agrees to the notion that OSS projects do violate Brooks’ law, but only because of the murky definition of ‘programmer’. He points out the fact that even though hundreds of people are involved in an OSS project, most of them are just fixing bugs and doing secondary work. “If you have a project that has five people who write 80% of the codes, and a hundred people who have contributed bug fixes or a few hundred lines of code here and there, is that a ‘105-programmer project”. So when someone says that OSS projects do break Brooks’ Law by showing that even when a hundred odd individuals help in the project by either finding bugs or contributing small pieces of code, they may still be missing the point that ultimately there is no change in the number of core developers involved in the project.

Nikolai Bezroukov in [5] gives another point of view that the non-applicability of the Brooks' Law is only applicable in cases of those open source projects where almost all architectural problems have been solved and a working prototype of the project has already been developed.

“Never in the field of human conflict has so much owed by so many to so few”

Winston Churchill

6. Leadership

The leadership structure of open source project teams is different from that of any other traditional organization. Usually there is a single leader or a committee of developers who are in charge of approving the patches and providing direction to the project. There will be other developers who are in charge of separate modules of the project and then the contributors to these modules. The structure thus formed places a lot of responsibility on the leader(s). Hence this section studies the leadership of open source project.

In “The Simple Economics of Open Source”, Josh Learner and Jean Tirole [15] identifies four points a leader needs to do in order to create a viable project and lead it through to success:

- Provide a vision

[15] states that a leader can provide a vision by assembling a critical mass of code and if possible a working prototype. Another thing he has to do is to make sure that there is enough work for the developers to catch their attention. Linus Torvald’s initial posting announcing the birth of Linux is an example of such a beginning. In the posting Linus did provide a significant amount of working code, but at the same time there were more than enough functionalities still to be developed, for anybody who was interested in, to step in and develop.

However, in [16], Kasper Edwards differs subtly by arguing that it is not the leader who provides the vision, but the software/code that the leader presents that offers the vision. To prove the argument, he references the Linux and Apache project. In both the cases, a major re-write of the code base was required, which introduced functionalities that were not evident from the initial statement of purpose as presented by the leaders. It was the vision of the developers that were mirrored in these changes. Based on this, Kasper states “It is the vision of what the co-developers wants from the software that triggers the programming effort”.

Whatever be the exact medium of providing the vision, it is clear that the leader has to make the initial contribution to the code base and the project should be challenging enough to attract enough developers.

- Divide project into smaller and well-defined tasks

A leader should try to make the structure of the project as modular as possible. This is a way to actively delegate work to others in an organised manner. However, Kapser asserts that this will just create more work for the leader. He feels that the leader of the project should not be responsible for defining modules and tasks. Rather, a leader has to only encourage the creation of modules and extra functionalities by designing the initial program in such a manner. Kasper feels that if the leader decides on the actual components and extra modules to be developed and then assign people to these modules, the “fun” of programming is lost and that developers will loose interest.

- Attract Programmers

Learner and Tirole are of the view that for a project to succeed, the leader should try to attract developers who can take up the implementation of various modules of the project. To attract these programmers, there should be a working piece of code as well as enough challenging undone work. The leader should try and communicate his vision for the project to try and attract the programmers.

As in the case of vision, Kasper is of the view that the leader per-say cannot attract the developers by just communicating the vision. A normal way in which a developer is hooked on to an open source project is as follows. He hears about the project either through email, bulletin board message or through website news. He then goes to the project website, reads as to what the project is trying to do and then downloads the code. He then goes through the codes and the present implementation. If he feels that there is something he wants done in that particular project or if he feels that the project is going to be a good challenge to him, he will volunteer as a developer. It is rarely the “vision” of the leader that attracts the developer. It is the code that speaks, not the leader.

- Keep the project together

Since the codes for an open source project is available for anybody to see and use, there is nothing preventing a dissatisfied developer from creating another version fork of the project.

This has happened in a lot of open source projects. BSD Linux is a project that resulted from the forking of the original AT&T Unix project.

The two factors that are pointed out in [15] that can prevent forking of OSS projects are trust in the project leader and loss of economics from a split. Thus a project leader can play a vital role in making sure that a project fork does not happen. He has to make sure that patches and contributions are not being delayed because of him becoming the bottleneck. However, at the same time he should not let the quality of the code base suffer by approving unwanted code contributions.

The developers should trust the objectives of the leader and should believe that the decision he makes is based of technical and design considerations and not based on ego driven, political or commercial issues. The leader should try to make the change management process as transparent as possible. In the case of Linux project, the merits of each contribution are discussed over emails and other medium and are available for anybody to follow. This makes it easier for the developer to trust the leader and the process involved.

An issue considered in [17] is the burnout of a project leader. The paper notes that due of the highly centralised structure that might evolve in the project, when the project becomes a success and scales beyond the capacities of a single person to coordinate. This might lead to continuous stress and final burnout of the leader(s).”Burnout is the price of becoming a media darling.” This has to be prevented. One way to do so is to make sure that trivial issues doe not have to go all the way to the leader, but be decided by relatively lower-level co-ordinators. Linux is an example. “My workload is lower because I don’t have to see the crazy ideas”, Torvalds said. “I see the end point of work done for a few month or even a year by other people”.

Kasper feels that it might be better to abandon the use of term “leader” in OSS development for a more correct usage, “maintainer”, since according to Kasper, that is what he is.

Summing up, a leader of an OSS project need to make sure that he provides a vision, attract programmers, make the project modular and prevent forks. At the same time he has to make sure the structure of the team is such that it can scale when the project becomes a success and he is not in danger of suffering from a burn out.

7. Conclusion

In this paper we looked at various aspects of open source software project management. We concentrated on issues related to requirements management, change management and leadership. We also looked at the Bazaar model of open source projects and studied how the bazaar model seems to outplay Brooks' Law.

We found out that open source projects are very informal in their requirement management process and rely heavily on informal mechanisms like email and bulletin board postings to power the process. Similarly, change management too is informal. We looked at some basic problems associated with the informal nature of the project management. Leadership and the role of a leader in OSS projects were also studied in the report. It was found that OSS leader have a high chance of burnout as the project grows and the administration process has to be streamlined to prevent this from happening.

We looked at the arguments that OSS projects seem to outplay Brooks' Law and found out that it generally is not the case. The appearance is brought about by the loose definition of a 'team" in these projects.

References

[1] ‘Open Source Initiative OSI’, http://www.opensource.org, 2002.

[2] Raymond, E., ‘The cathedral and the bazaar: musing on Linux and Open Source by an accidental revolutionary’, O’Reilly and Associates, Sebastopol, 2002.

[3] ‘Fetchmail’, http://www.tuxedo.org/~esr/fetchmail/, 2002.

[4] Asklund, U., Bendix, L., ‘A study on configuration management in open source software projects’, IEE Proceedings – Software, Vol. 149, No. 1, pp 40 – 46, February 2002.

[5] Nikolai Berroukov, ‘A second look at the cathedral and the Bazaar’, First Monday, Vol. 4, No. 12, December 1999. http://firstmonday.org/issues/issue4_12/bezroukov/

[6] Scacchi, W., ‘Understanding the requirements for developing open source software systems’, IEE Proceedings – Software, Vol. 149, No 1, February 2002.

[7] Nuseibeh, R., Easterbrook, S., ‘Requirements engineering: a roadmap’, ACM and IEEE Computer Society Press, 2000.

[8] Berliner, B., ‘CVSII: parallezing software development’, Proceedings of USENIX Winter, 1990.

[9] Linus Torvalds, ‘Re: [PATCH] Remove Bitkeeper documentation from Linux tree’, http://lwn.net/2002/0425/a/ideology-sucks.php3, April 2002.

[10] ‘BitKeeper’, http://www.bitkeeper.com, 2002.

[11] Feldman, S.I., ‘Make: a program for maintaining computer programs’, Software – practise and experience, 1979.

[12] Brooks, Frederick P., ‘The Mythical Man Month: Essays on Software Engineering’, Addison-Wesley Pub Co, 1974.

[13] Jones, P., ‘Brook's Law: The more the merrier?’, IBM Developer Works, 2000. ftp://www6.software.ibm.com/software/developer/library/merrier.pdf

[14] ‘Free Software Foundation’, http://www.fsf.org, 2002.

[15] Lerner, J., Tirole, J. "The Simple Economics of Open Source," Working Paper 7600, National Bureau of Economic Research, 2000.

[16] Edwards, K., ‘When Beggars becomes choosers’, First Monday, Vol. 5, No. 10, October 2000. http://firstmonday.org/issues/issue5_10/edwards/

[17] Bezroukov, N. "Open Source Software as a Special Type of Academic Research," First Monday, Vol. 4, No, 10, 1999. http://firstmonday.org/issues/issue4_10/bezroukov/