Open Source and Closed Source Software Development Methodologies

Vidyasagar Potdar, Elizabeth Chang
School of Information System, Curtin University of Technology, Perth, Australia 6845
PotdarV@cbs.curtin.edu.au, ChangE@cbs.curtin.edu.au

Abstract

Open source software development represents a fundamentally new concept in the field of software engineering. Open source development and delivery occurs over the Internet. Developers are not confined to a geographic area. They work voluntarily on a project of their choice. As new requirements emerge, the software is enhanced by the user/developers. In this paper we show a comparative study of open source and closed source software development approaches and present a software life cycle model for open source software development.

1. Introduction

The concept of free software is not new. It has been around since the 1960s in universities such as MIT and corporate firms such as Bell Labs who freely used source code for research [1,2,3,4]. Software was not a means of revenue generation but was used to hook more and more customers to buy new computers [5]. In the early 1980s, Microsoft started writing software for the sole purpose of profit. It gave only compiled code; source code was hidden from the user. This move had a great impact and could be considered as the birth of open source. Richard Stallman, researcher at MIT, founded the 'Free Software Foundation' (FSF) to develop and distribute software under the General Public License history (GPL) while Bruce Perens defined a set of guidelines that a software license must grant its user, and he called this Open Source Initiative (OSI). In this paper, we briefly describe the open source software development and compare it with closed source software development. The paper is organized as follows; in section 2, we describe how open source software is developed. In section 3, we compare open source and closed source software development approaches. In section 4, we propose a life cycle model for open source software development.

2. Open Source and Who, Why, What?

Bruce Perens defines that Open Source is a specification of what is permissible in a software license for that software to be referred to as Open Source [2].

2.1 Who is an open source developer?

Simply put, "any one who contributes to the open source project is an open source developer", such as a user of the software, a developer who develops the software, a debugger or hobbyist who likes spending time on open source, or a promoter who funds such a development.

2.2 Why do they produce open source?

Eric states that developers are attracted towards open source development because that gives them an opportunity to demonstrate their ability. So they voluntarily select a project and start contributing. When programmer's, code gets accepted, it boosts their ego and they get recognized for their effort in the community, [7] Peer recognition creates reputation and a reputation as a good programmer is a great achievement.

2.3 What do they do in open source?

Open source developers are involved in a variety of activities such as designing, coding, debugging and utilizing. Each activity occurs simultaneously. Parallel development and debugging is the key to open source success. Users also play a vital role in the debugging process by reporting bugs to developers or sometimes fixing it themselves. Developers are well aware that users are the best beta testers [7,8].

2.4 What are the major customer concerns?

2.4.1. Maintaining consistent software architecture. This is one major concern, but it's outside the scope of this paper [12, 13].

2.4.2. Support and coordination for deployment. When end users utilize open source component they want to know whether the component is fully automated or does it need any integration with other components. However when end users come up with bug reports, feature

requests or installation guidelines they expect a full support and coordination from the developers.

2.4.2. Managing upgrade and complexity. The end users are concerned with the future upgrades. They are not sure, as components get more and more powerful and large, whether the complexity of the code can be automatically managed. Managing complexity is a big concern because OSS often has parallel developments going on [1].

3. Open Source versus Closed Source Software Development

Open Source Software Development (OSSD) is a recent phenomenon, while traditional closed source software development (CSSD) has been here since the dawn of software development. One major difference between these two models is source code visibility. In this section we will point out most of the differences between these approaches. We begin with their process models.

3.1. Process Models

CSSD normally follows a spiral or iterative model of development i.e. software development goes through all phases like planning, design, implementation [14] whereas OSSD follows an evolutionary model for development where the software never reaches a final state and keeps on evolving according to customer needs [15]. It's more of a concurrent or parallel process. CSSD has clear cut milestones using which the progress of the project can be tracked but in case of OSSD it not possible although CVS do help to keep some track.

3.2. Requirements Definition and Specification

CSSD starts with requirements definition and specification. Here requirements are vague. Project developers are not aware of the actual requirements. They need to interview stakeholders to elicit requirements and then start implementing [1, 14]. On the contrary OSSD starts with a motive of requirements satisfaction. Requirements are clear, as developer is fully aware of the requirements [15]. In case of CSSD all the user requirements may not be implemented because of time or budget constraints [16, 14] where as in OSSD it possible because user is often the developer [1, 15]. In case of CSSD system architect and project manager decide which requirements will be incorporated while core members of the open source project decided which requirements to be implemented [14].

3.3. Documentation

In case of CSSD project plan is document and followed. Once the requirements are clear they are documented [14]. Even the designing and testing procedure is documented. Where as in OSSD there may or may not be any official documentation. [14,16]

3.4. Analysis and Design

In CSSD, system architects and project mangers spend a lot of time in designing the project [14], whereas in OSSD designing is often merged with implementation.[17, 12, 15].

3.5. Software Architecture

Maintaining consistent software architecture is enforced during the development phase itself, there is rarely a drift between software's conceptual and concrete architecture. [12, 13, 14]. In case of OSSD this has been recognized as a major concern. Maintaining consistent software architecture is difficult because of its highly collaborative and distributive nature. [12, 13]

3.6. Implementation

Only one implementation is possible for one requirement [14] whereas in case of OSSD multiple implementations are possible for the same requirement. This is considered as a major issue as it results in code forking. The rate of development is comparatively slower that the open source because the number of developers assigned to a CSS project can never match a full-scale open source project like Linux [6, 7, 14]

3.7. Source Code

In CSSD source code is hidden from the user while in OSSD source code is open as a result user can view and modify the code to suit individual needs [2, 15]. Such freedom is not available in closed source software.

3.8. Testing

In OSSD, users act as bug reporters and beta testers. Whenever a user finds any bug in the software they either try to solve it or bring it to the notice of the community [1]. But commercial closed source products use service packs to repair bugs [15]. OSS community believes "*no bug can survive wide testing*" [7].

3.9. Release and Delivery

Open source products are released quite often on a daily or weekly basis whereas closed source products are released on a yearly basis. In commercial softwares,

product is often released due to marketing pressure and tight schedule whereas open source products are released once the developer thinks that it has reached a functional stage [1, 14].

3.10. Maintenance

Service packs are needed quite often to repair bugs in commercial closed source products, whereas bug reporting and bug fixing takes care of maintenance in case of open source products.

3.11. Product

Close source products may soon reach a finalized state once the documented requirements are implemented whereas open source products are always in a evolutionary phase because as requirements emerge they get implemented [14].

3.12 Type of Software

Commercial software development is more of a solution kind of development. Developers create solutions for a big company. It is more like customized development whereas open source development is more components-based i.e. plug-n-play type software. Developers create small programs, which work on a variety of platforms.

3.13. Security

It is a common belief that commercial closed source product is highly secure because it is developed by a group of professionals confined to one geographical area under a strict time schedule. But quite often this is not the case, hiding information doesn't make it secure, it only veils weaknesses [10]. In CSS security is achieved through obscurity, however in OSS security is achieved through 'open source'. The ability to modify the source code works to your advantage if you want to deploy a highly secure system. One can ask for a third party security certificate or get the system scrutinized by a professional security expert for possible back door entries. [10, 11], such freedom is not available with commercial closed source products. Another argument that supports open source security is community reaction to bugs. Community reaction to bug reports is much faster compared to commercial closed source which makes it easier to fix bugs and make the component highly secure.

3.14. Productivity, Quality and Cost

Developer's productivity may decrease if they are forced to work on a project in which they are not interested, which is contrary to OSSD where developer is free to choose the project on which they want to work [15]. Developing open source software is faster, better and cheaper. All the three factors can be satisfied simultaneously. Cost is reduced because no one is paid for the job everyone is a volunteer. Speed is increased because development is parallel and collaborative in nature. And finally quality is maintained because the product is released only when the developer think the product is stable and functional [16]. However with CSSD this doesn't work well. At one time only one factor can be satisfied fully. E.g. if speed is maintained quality and cost may go up or if cost is to be maintained quality and speed may go down [9, 14, 16] Hence CSSD can be considered slow and expensive.

3.15. Environment

Often we find centralised, single site development in CSS while decentralised, distributed, multi-site development in OSS. In CSS development happens in a geographically confined area, while in OSS development occurs on the Internet [1, 14].

3.16. Group work and Communication

Open source is co-operative and need high level of co-ordination over the Internet and multi-site. Lack of coordination among developers results in code forking [1]. However in case of CSS, inconsistency is easily managed by face to face or weekly team meeting.

4. New Life Cycle Model for Open Source

Software life cycle determines the set of activities that constitute a software project. It may not be surprising that the OSSD life cycle differs from CSSD [15].

4.1. Traditional life cycles do not suit open source

In case of waterfall model, after every stage documentation is done. But in case of OSSD often simultaneous development occurs. OSSD has more of a modular approach. Several individual developers can simultaneously work on several different modules without worrying about the final integration. This is one of the reasons why waterfall model doesn't suit OSSD. Waterfall model is linear in nature. It has clear milestones. But OSS has very vague milestones. Because of its evolutionary nature it never reaches a stage where we can confidently say it has finished a particular phase. As OSSD has vague milestones it's really difficult to point out its progress. This is another reason why we feel the existing waterfall model is not suitable for OSSD.

4.2. Some phases in traditional software life cycle do not apply to open source

Commercial CSSD is confined to a geographic location. During planning stage project leader organizes teams, schedules meetings, and assigns roles and responsibilities to the team members. When we compare this with OSSD we figure out that there are no teams, no meetings at all as all the communication happens through mailing lists or project web sites. User or developer is free to choose a project of their own choice. This removes the need of assigning roles to developers.

In CSSD planning is followed by requirement elicitation, requirement documentation and feasibility analysis. While in case of OSSD the requirements are not vague, they are clear to the developer, as most often a user is also a developer. Apart from that, open source development sites like www.sourceforge.net, www.kde.org have a feature where users can request new features to be implemented in particular software, which reduces the need for requirement elicitation. Since the requirements are clear to the developer they just need to be analysed and implemented. This removes the requirement elicitation phase from OSSD. Design is generally followed by implementation. Since there is a large list of requirements, normally each and every requirement does not need detailed design. In such cases requirements are directly implemented. So we remove the detailed design phase from OSSD.

Multiple implementations create complexity and raise the issues for proper coordination. Traditional life cycle didn't tackle this problem but this is a phase which we think should be introduced in open source SDLC.

Concurrent implementations create architectural defects. As the user requirements change, the software changes to fulfill these requirements. Often these changes are done without considering the conceptual architecture. As changes go on adding up; the software's concrete architecture starts drifting from its conceptual architecture. OSSD is highly prone to this because of its collaborative and distributive nature. Hence we need a phase to tackle this issue as well [12, 13].

OSS lacks coordination and hence it is sometimes difficult to manage. This necessitates the need for a separate or simultaneous phase to manage complexity and maintain coordination. This phase is normally not taken care off in traditional life cycle [1].

We hope it is clear from the above discussion that there is a necessity for some phases while others are not required. The issues outlined above need to be considered when a SDLC model for open source is proposed.

We propose a requirement oriented pendulum model to solve some of the issues highlighted above. The proposed model tries to provide guidelines on how open source project should proceed. Our main aim is to try and estimate the progress of an open source project.

5. Requirement Oriented Pendulum Model for Open Source Software Development

We propose the requirement oriented pendulum model, to provide a mechanism to track which requirements have been submitted and which of those have been implemented and tested. The proposed model has four stages which we described below.

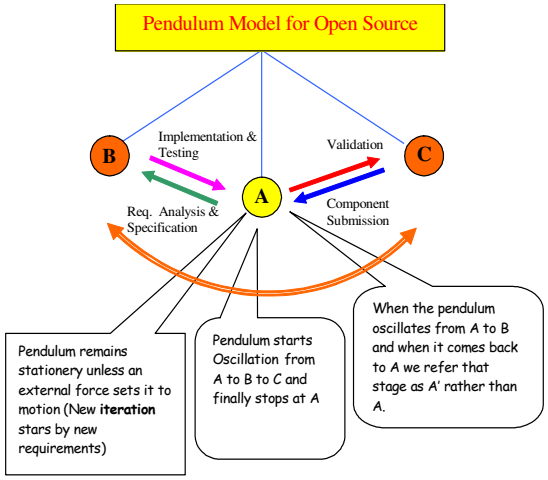

Figure 1: The Pendulum model for Open Source

5.1. Requirement Analysis and Specification.

In the pendulum model this is the first stage in which requirement analysis is done. Different users in the system have different requirements. Users submit feature requests which become new requirements. At this stage the pendulum is in the center. When the requirement analysis and specification starts the pendulum gets into motion. Once requirements are analyzed it reaches the left most end, which is represented by state B.

Initial State	Oscillation from	Reason for transition	Final State
A	A to B	Req. Analysis & Spec.	B
B	B to A'	Implementation & Testing	A'
A'	A' to C	Validation	C
C	C to A	Component Submission	A

5.2. Implementation and Testing

Two main activities in this phase are implementation and testing. Major outcome of this stage is source code. Coding involves writing source code to implement a requirement. Once the requirement is implemented it should be tested. So in the second half of this phase testing should be conducted. It requires careful planning and coordination. Major goal should be to make sure all components created during implementation function properly. Apart from that, this stage should include alpha

testing and beta testing. The testing should start with standard test cases. Each test case should contain information like test case name, action to be performed, data to be used, expected outcomes, actual outcomes. Normally this is done by the users or by the developers. When the implementation and testing starts the pendulum moves from B to A'. It reaches the centre, which is represented by A'.

5.3. Validation

This is the next iterative phase in the pendulum model. Major goal of this phase is to carry out field testing and validation and then generate reports. Bug reports are a major result of validation. Bug reporting should be done by user or developers. This can be done in many ways, but it is suggested that one should use a bug reporting software like Bugzilla or emailing the bug reports in a pre-specified format to the project leader. However before reporting a bug one should make sure that someone has not reported it earlier. When validation starts, the pendulum moves from A' to C. It reaches the other end, which is represented by C.

5.4. Component Submission

This is the final phase of the pendulum model. This phase has two activities submitting new feature requests and submitting tested and validated components. Once the validated components are submitted a new version of open source component is developed. At the same time new feature request which are submitted by the users or the developers sets the pendulum into motion once again. Feature request becomes the basis of new requirement and as discussed above sets the pendulum in motion. When component submission starts, the pendulum moves from C to A. It reaches the initial position once again. This is when one full oscillation completes.

6. Conclusion

From the study that we have conducted, it has come to our notice that OSSD is similar to its traditional counterpart in many aspects, but there are many areas in which it differs tremendously and these features make it different from the CSSD. As a concluding remark, we can say open source software is a competent alternative to CSS and the pendulum model that we proposed can very well demonstrate the functioning of OSSD.

7. References

[1] Steven Webber, "The Political Economy of Open Source Software", California, June 2000.

[2] Bruce Perens, "The Open Source Definition" Available at: http://perens.com/Articles OSD.html Acc. on: Oct 2002

[3] Malcolm M. "Profit Motive Splits Open Source Movement", Aug 26th, 1998. Available at http://content.techweb.com/wire/story/TWB19980824S0012 Acc. on: Oct 2002

[4] "What is Free Software Foundation"? Available at: http://www.gnu.org/fsf/fsf.html Acc. on: Oct 2002

[5] "Overview of GNU Project" Available at: http://gnu.j1b.org/gnu/gnu-history.html Acc. on: Oct 2002

[6]Ko Kuwabara, "The Bazaar at the Edge of Chaos" Chap 2: A Brief History of Linux. December 1999. Available at: http://www.cukezone.com/kk49/linux/chapter2.html. Acc. on: Oct 2002

[7] Raymond, E.S., "The Cathedral and the Bazaar" O'Reilly & Associates, 2000.

[8] Vinod Valloppillil, "Open Source Software, A (new?) Development Methodology" Nov 1998.

[9] Lerner, Joshua and Tirole, Jean, "The Simple Economics of Open Source" (February 2000). http://ssrn.com/abstract=224008

[10] Ferrara Linux User Group, "Open Source and Security" 2001 Available at: http://members.ferrara.linux.it/pioppo/aeronautica2001/opensecurity-2x1.pdf Acc. on: Nov 2002

[11] Dare Obasanjo, "The Myth of Open Source Security Revisited v2.0", 2002. Available online: http://softwaredev.earthweb.com/sdopen/article/0,,12077_990711,00.html Acc. on: Nov 2002

[12] Tran, J.B., Holt, R.C., "Forward and Reverse Architecture Repair" *Proc. Of CASCON '99*, Toronto, pg 15-24, 1999

[13] Tran, J.B., Godfrey,M.W., Lee, H. S., Holt, R.C., Architectural Repair of Open Source Software. *In Proceedings of International Workshop on Program Comprehension*, Limerick, Ireland, June 2000.

[14] Satzinger, Jackson, Burd "System Analysis and Design in a Changing World", Thomson Learning, 2000.

[15] Scott H, Charles W, Plakosh D., Jayatirtha A., "Perspectives on Open Source Software", Software Engineering Institute, Pittsburgh, Nov 2001 p49.

[16] Walt Scacchi, "Is Open Source Software Development Faster, Better and Cheaper than Software Engineering?", *23rd International Conference on Software Engineering*, Toronto, Ontario, Canada, 2001.

[17] Tran, J.B., "Software Architecture Repair as a Form of Preventive Maintenance", Masters Thesis, University of Waterloo, 1999.