Study and Analysis of The Team Structures on Development Processes

Er. Amandeep Kaur  
Asstt. Prof., CSE Deptt.  
GIMET, Amritsar

Er. Satish Kumar  
Asstt. Pof., IT Deptt.,  
GIMET, Amritsar

Er. Satnam Kaur  
ASE at TCS

Abstract
In recent years open source software has seen magnificent success as it produces high quality software at lower cost than the traditional commercial development. Open source proves itself highly suited, both as software product and as development methodology. This paper investigates the benefits as well as the limitations of the open source development and makes an effort to discuss the open source project community structure in relation to the organizational community structure. The focus of this paper is to examine the inner workings of the development process of successful and mature Apache and Mozilla foundation and provide the analysis of the role hierarchy, role migration, development community size, work distribution, problem reports and code ownership through comparative case studies of Apache and Mozilla.

Keywords

INTRODUCTION
Proprietary or closed software is software whose source code is not made public but the copies of binary are made public and are owned by a company or individual. The term free software was coined by Richard Stallman in 1984 which has an unintended meaning, Software one can get for zero prices, as well as the software which gives the user certain freedoms. The term Open Source Software was coined by Eric. S. Raymond in 1998. Open source refers to a program whose source code is freely and publically available for use and modification from its original design. Open source software is computer software whose source code is freely available and gives the freedom to view, download, modify, and redistribute the code through the specific licensing agreements vary as to what one is allowed to do with that code [10]. Open Source Software refers to software whose license gives the users four basic freedoms:

- To run the program for any purpose.
- To study the workings of the program, and modify the program to suit specific needs.
- To redistribute copies of the program at no charge or for a fee.
- To improve the program, and release the improved, modified version [10].

The closed source development is like the building like a cathedral with central planning, tight organization and one process from start to finish where hierarchical structures exist and little collaboration takes place. The proprietary software applications are strictly protected through patents and intellectual property rights, The free software movement is working toward the goal of making all software free of intellectual property restrictions which hamper technical improvement [18]. Free software is a social movement which is based on shared values. The open source development is like a great babbling bazaar with loosely centralized cooperative community where people freely contribute to the development of software through collaboration. Open source software is developed by an on-going, iterative process which starts with problem discovery and is completed with the help of large community of developers and users [18]. Open source software does not pay royalties as no copyright exists. Open source software is not a social movement, but is a method of developing the software.

LIMITATIONS OF TRADITIONAL APPROACH
The traditional software development projects suffer from various problems. These projects are prone to time and cost overruns, are largely un-maintainable, with questionable quality and reliability. About 75% of software projects fail in one or more of these measures and other are cancelled due to failure as revealed by the 1999 Standish Group report. In addition, these projects fail to satisfy the needs of the customer for whom they are developed. [18]

The major failures in traditional software development are as follows:

- In adequate understanding of the size and complexity of development projects coupled with inflexible, unrealistic timeframes and poor cost estimates.
- Lack of user involvement.
- Shortfalls in skilled personnel: Team members with insufficient technical expertise, managerial skill or knowledge about the problem domain can affect project success.
- Project costs are further increased by the price of license fees for software and tools required for application development as well as add-on costs for exchange controls.
BENEFITS OF OPEN SOURCE SOFTWARE

In order to remove the difficulties of traditional approach, open source software systems are used. Open source software produces reliable, high quality software in less time and with less cost than traditional methods.

- **Quality software** - As the collaborative development allows multiple solutions, open source software features produces very high quality software.

- **Development speed** - The speed of developing the open source software is high due to the reuse of source code, and many people contribute to the code development. As the more people are creating code and adding value to a project, the quicker the product is released and becomes valuable to a user group.

- **User involvement** - Open source software is highly benefited by user involvement. Users are treated as a valued asset in the development process. Viewing users as co-developers leads to code improvement and effective debugging. Users can also assist developers in finding system faults and improvements, thereby reducing the need for extra developers to perform the same function.

- **Access to existing code** - A huge amount of open source project code and open source toolset can be accessed by the developers which can speed up the development.

- **Collaboration** - Open source software consist of large pool of highly talented and globally dispersed professionals who follows the collaborative approach to problem solving through constant feedback and peer review. There is also the collaborative and parallel development involving the source code and reuse. These professionals participate in open source projects for improving skills as well as for financial reward [18].

- **Redistribution** - The open source software can be redistributed and the license does not restrict any party from giving away or selling the software. No royalty or fee is required for such sale [11].

- **Take control of software** - In proprietary software, the control on source code, budget and up gradation is in the hands of vendor. But in open source software, one has the right to control his/her own code, power to modify the code according to his/her own requirements, control the budget, control on up-gradation process and can choose to decide which bugs to fix and when to fix.

- **Cost** - In open source software, source code is free and available to any user to modify and improve. Testing of open source software can be done free of cost. Once acquired, there are no license fees for open source software, reducing the annual license fees cost to zero. The cost of sale is also zero, because no additional license is required as the installation grows. Only the cost is spent towards the support of our software which ensures reliability and reduces the running cost.

- **Continuity** - In proprietary software, company may decide to discontinue the service for software. In open source software, the software is not dependent on a single entity and it gives the users the advantage of the community. Software may be picked by any of the developer who can continue to its development from where it was stopped.

- **Scaling and consolidating** - Open source software can be easily scaled. With load balancing and open source applications such as email and database, one can enable their organization to either scale up or achieve higher growth.

- **Outsourcing services to o2i** - The open source services can be outsourced to o2i which can provide the perfect open source software solutions that effectively meets the requirements of an organization [32].

LIMITATIONS OF OPEN SOURCE SOFTWARE

Open source software is flexible in nature, but they suffer from certain limitations also. These limitations arise from the fact that OSS is often developed and supported by communities of volunteer programmers who create and modify the source code. These limitations are as follows:

- **Security** - Open source software suffers from security issues. Trojans can capture private and confidential information without the user ever knowing.

- **Usability** - The technology is not reviewed by usability experts, and does not cater to vast majority of computer users. Therefore it has been highly criticized for its lack of usability. System deployment and training is often more expensive with open source software [19].

- **No guarantee of development** - It is not possible to know whether the project ever reach a usable stage, and even if it reaches, it may die later if there is not enough interest. Therefore, there is no guarantee that development will happen.
Intellectual property - There are many problems which are connected to the intellectual property. Many of the countries are now accepting the algorithm and software patents, but it is very difficult to know that a particular method used to solve a software problem is patented. So, the community can be considered guilty of intellectual property.

Difficulty in knowing the existence - Open source software, especially which are not backed by a company willing to invest resources is not advertised too much. Several aggregation points for open source software does not exist which makes it difficult to know that a project exists, and its current status [3].

Technical support - Open source software is not really owned by an individual or an organization; it does not guarantee high level of technical and general support and services. Technical support means a user who is experiencing difficulty posts a query to an online OSS forum and then waits for a member of that forum to read the posting and reply. Problem resolution is done through community discussion and testing. The main limitation of such an informal support system is that the answers are not readily available, and the users has to wait for a long time ranging from seconds to days which put the major effect on the usability of software.

Forking code - Open source software suffers from the problem of forking code. Open source software is open for the users to modify it and it is also easy for the individuals to use the same source code to develop different non-compatible software. The open source software programmer may take a different fork and with each different fork, two programs that were identical become different from one another. Forking code causes various problems like the generating of software which is not compatible for others to use, software does not work in desired way, users become frustrated with abandoning software, etc [10].

Speed - Due to the absence of formal management structures, the results could be slower in the rapid development environment.

Strong user involvement - Strong user involvement and participation throughout a project is becoming problematic as users tend to create bureaucracies which hamper development.

User interface - The user interfaces of open source software are not very intuitive [4].

RELATED WORK

(Mokus et al, 2002) [1] has studied and analyzed the various aspects of the Apache web server and the Mozilla browser and compared them with the commercial projects. They found that the development of Apache was decentralized with only about 15 developers contributing more than 80 percent of the code for new functionality and the bug reporting was quite decentralized with the top 15 reporters submitting only 5 percent of problem reports. On the basis of the analysis of Mozilla data and apache data, they came up with several hypotheses and also provided the description of the development process. On the other hand, (Yunwen Ye et al, 2003) [15] has analyzed the GIMP project in order to understand the motivation behind the new members joining and aspiring to have more influential roles in an open source project. They found that one of the motivational forces is learning which is possible through Legitimate Peripheral Participation (LPP) and is supported by analyzing the social structure of OSS communities and co-evolution between OSS systems and communities. Finally they have noticed that there is a relationship between active participation in the mailing list and the contribution made to GIMP software, and the GIMP community is a meritocracy. (Kevin Crowston et al, 2003) [12] has analyzed the bug trackers for 120 open source projects from SourceForge and studied the social communication structure in the projects. They find that a consistent core-periphery shift pattern does not exist across different projects. (Jin Xu et al, 2004) [6] has studied the role of developers and users in the open source software community, and their activities which helps in determining the development of projects. The Project leaders guide the vision and direction of a project. The peripheral developers irregularly where as the central developers regularly fix bugs, add features, submit patches, provide support, write documents and exchange other information. Core developers manage CVS releases and coordinate peripheral developers and central developers. Passive users download the code and use it for their needs. Active users discover and report bugs, and suggest new features. It also includes the study of the whole developer community at SourceForge in order to perform the quantitative analysis of open source software developers. (IL-Horn Hann et al, 2004) [13] has explained the reason for participating in the open source software development, explored whether the participation is consistent with the well-established theories from labor economics and established two plausible theoretical bases for the existence of returns to open source participation; viz., human capital theory and signaling theory. It is difficult to assess the performance of programmers and hence changes in their human capital in the case of Apache project. In Apache project, open source community effectively screens programmers based on their productive capacity. (Jin Xu et al, 2005) [8] has studied the open source software development community at SourceForge and performed the quantitative analysis of open source software developers on the basis of SourceForge 2003 data dump. Statistics and social network properties are explored from which they found the different degree distributions in small and large projects. Co-developers and active users are main part of large projects whereas project leaders and core developers are the main part of small projects. Topological analysis on four subsets of OSS development community is also conducted in order to identify the different characteristics of each subset and determine the effect of different community member. (James M. Bieman et al, 2005) [16] has gathered enough data in order to examine that whether the FreeBSD...
supported the six hypotheses proposed by Mokus et al. Comparative analysis with the Apache and Mozilla open source software project is also conducted. They found that only 3 hypotheses: relationship between the core developers, developers and contributors and defect density of OSS are supported, core developers implement smaller percentage of system, more than 15 developers contribute 80 percent of codebase and the testing process is better defined. (Kevin Crowston et al, 2006)\(^{[11]}\) has performed the social network analysis of large number of projects in order to determine the level of centralization and hierarchy in open source software projects. The analysis indicated that all of the OSS projects are neither uniformly centralized nor hierarchical. The centralization in OSS projects is distributed. A few of the OSS projects are highly centralized, few decentralized and most of the OSS projects lie somewhere in middle. Hierarchy in OSS projects is not static. It develops and evolves over time. (Ju Long, 2006)\(^{[9]}\) has investigated the sustainability of the open source software development model and distinguished the core developers from non-core developers in the community. Both core developer leadership and project advocating activities are crucial in the development of open source software projects. Open source developers can learn the role of promoting and publicizing the projects in OSS community. The releasing of new versions of projects is paying more attention in motivating the developers in the community. (Peter C. Rigby et al, 2006)\(^{[14]}\) has studied the code review process used by 11 open source projects, discussed the four open source projects including the Mozilla and Apache server project and found similarities in the request for small and complete patches and differences in the community policy appeared to dictate the level of patch review. Some other commonalities in the review process: each project had a coding standard; projects required contributors to update documentation; and they emphasized that patches should be separate and no patch should add large functionality, because small patches are easier to review are also found. The developer and commit mailing lists are also extracted into a database in order to reconstruct and understand the review and patch processes of the Apache project. (Walt Scacchi et al, 2007)\(^{[19]}\) has analyzed the role migration and project advancement process, and role-sets by examining and modeling the processes through comparative case studies within three large OSSD projects that embed the web software infrastructure. These projects include Mozilla.org, Apache.org, and NetBeans.org. Role migration and project career advancement processes helps in determining how socio-technical processes are structured within conventional software development processes. Overall, they found that comparative study of socio-technical processes provide sufficient substance and detail to reveal the richness of processes, practices, and roles that shape open source software development projects. (Kevin Crowston et al, 2008)\(^{[8]}\) has investigated the structure and the coordination practices adopted within four FLOSS development teams and analyzed the bug fixing process which is considered one of the main areas of FLOSS project effectiveness. They have found that there are striking differences in the level of contribution to the open source development process. The most active users carried out most of the tasks while most others contributed only once or twice. The accomplished tasks, the adopted coordination mechanisms, and the role undertaken by both the FLOSS development team and the FLOSS community has been identified on the basis of the messages recorded in the bug tracking system of four projects. (Mehrdad Nurolahzade et al, 2009)\(^{[15]}\) has explained the patch evolution process of the Mozilla development community and performed the quantitative analysis in order to examine the inner-workings of the process of Mozilla foundation. Mozilla development community includes developers, module peers, module owners, bugzilla component owners, smoke test coordinator, reviewer and super reviewer. They have highlighted how different parties involved affect the process and addressed the various questions like which types of bug reports attract more community participation, which bug reports receive no community attention and what type of bug reports are resolved without receiving any community attention beyond the assigned developer and the module owner. (Chintan Amrit et al, 2010)\(^{[13]}\) has analyzed the Socio-Technical Core-Periphery structure of open source projects which can give the open source project leader and the community a better understanding of who is working on which part of the software at any given point of time. They firstly borrowed the concept of Core-Periphery from social network field, and applied it to software call graph; then they determined the open source software repository to determine which developer is working on Core or the Periphery of the software call graph, at any given point of time in order to arrive at the Socio-Technical Core-Periphery structure. Sufficient information on the Socio-Technical health of a particular project can be obtained by integrating such information into open source project portals such as SourceForge.

COMMUNITY STRUCTURE AND ITS IMPORTANCE

The community structure of open source software projects can be shown by onion model in which the individuals who have more interactions are more central than those who have less. The role closer to the center has a larger radius of influence than others i.e., the central individuals are more active. The open source software development team consists of small team of core developers who are responsible for contributing most of the code and for overseeing the development, a large group of codevelopers who contribute by reviewing or modifying code, fixing bugs, and a large group of active users who contribute bug reports not the code.

The Project Leader has larger influence than that of a Core Member, who in turn has a larger influence than an active developer, and so on. There is least influence of the passive users, but they still play important roles by attracting and motivating the other members.

The structure of open source software community is unique which depend upon the nature of the system and its member population. The OSS community structure differs in the percentage of each role in the whole community. In general, most members are Passive Users. For example, about 99% of people who use Apache are Passive Users. The percentage drops sharply from Readers to Core Members. Most systems are developed by a small number of developers.
Most of the open source software is developed by the distributed teams who rarely or never meet face to face and coordinate their activities by the means of emails and bulletin boards. The open source software development has communication and development centralization. The development centralization refers to the writing of code and the communication centralization refers to the communications between project members that are found in email, bug reporting system and instant messaging.

The social structure of OSS projects is useful in risk management and the better understanding of the social structure can improve the development planning.

A. OPEN SOURCE SOFTWARE COMMUNITY ROLE

An OSS development community is composed of groups of loosely connected contributors with central coordinators and decision makers. Open source software community can be classified into two groups as User group and Developer group as shown in figure 1.

![figure 1: Open Source Software Community](image)

- **User group**: The user group includes passive users and active users. These can be defined as follows:
  - **Passive users**: Passive users do not directly contribute in the development of the software projects. They generally use the system like any other commercial system. The high quality and the potential of being changed when needed are the main factors that attract the passive users towards the OSS.
  - **Active users**: Active users not only use the system, but also try to understand how the system works by reading the source code. They can suggest new features, discover and report bugs and exchange other useful information by posting messages to forums and mailing lists. Different users read the system for different purposes. They have some contributions except modifying code. Some read the system to learn programming while another read an OSS system not for the purpose of improving the system but for understanding its underlying model and then using the model and reference model to implement similar systems.

- **Developer group**: The developer group can be classified as follows:
  - **Project leader**: Project Leader is the person who initiates the project and is ultimately responsible for the vision and overall direction of the project.
  - **Peripheral developer**: Peripheral developers occasionally contribute new features or functionality to existing system. They irregularly fix bugs, submit patches, provide support, write documents and exchange other information, i.e. their contribution is irregular and period of involvement is short.
  - **Active or central developer**: Central Developers are the major development force of OSS systems. They regularly fix bugs, add new features, submit patches, provide support, write documents, and exchange information.
  - **Core developer**: Core developers are responsible for guiding and coordinating the development of an OSS project. Core developers are involved with the project for a long time and made significant contributions to the development and evolution of the system. They extensively contribute to projects; manage CVS releases and peripheral and central developers. They can also be called as maintainers in some communities.
  - **Bug fixer**: Bug fixer fix bugs that either they discover by themselves or are reported by other members. They have to read and understand a small portion of the source code of the system where bug occurs.
Bug reporter - Bug reporter discovers and report bugs; they do not fix the bugs themselves and they may not read source code either. They assume the same role as testers of the traditional software development model. The existence of many Bug Reporters assures the high quality of OSS, because “given enough eyeballs, all bugs are shallow.

B. ORGANIZATIONAL COMMUNITY STRUCTURE

The open source software community can be organized into two distinct groups called as core developers and non-core developers (community developers). In order to distinguish between the roles of core developers and community developers, one must have the clear understanding of the organizational structure of OSS projects. Let us have the brief description of the dimensions of organizational structure for both the community and core developers shown in figure 2.

- **Coordination mechanisms**: - The coordination mechanisms in the traditional organizations are based on hierarchical supervision, plans and procedures. In OSS development process, the coordination mechanisms are highly decentralized and democratic. To coordinate development activities, the community developers use online forums as the essential channel whereas the core developers use email lists as an essential channel to communicate with each other. Many development tasks are assigned to community developers, and many development decisions are made through a forum. The core developers may also use concurrent version system (CVS) as another important channel to coordinate the development activities. The total number of CVS updates could indicate how active the core developers are.

![Figure 2: Dimensions of organizational structure](image)

- **Distribution of decision making authority** - The distribution of decision making authority in traditional organizations is highly centralized. There is no single central decision maker in the OSS development process. The community developers make judgments on what tasks to do and how to do them.

The OSS project almost always starts with a single developer. If the project starts to grow after the first couple of releases, it will attract more and more developers who will voluntarily join the project. However, that does not mean that there is no centralized decision making. The core developer is responsible for setting the goal and mission of the project, and for dividing the project into different tasks. The coordination among these tasks is also the responsibility of core developers. One or two core developers cannot handle all these activities. More core developers could mean more talents and resources to fulfill these responsibilities.

- **Division of labor and development process**: - The division of labor in the traditional organizations is decided by inputs and outputs analysis. In OSS projects, the division of labor is drastically different as the large numbers of developers volunteer to develop the project. These developers are organized in loosely centralized and networked community. No formal development plan or strict schedule is followed, and the developers choose in which projects to participate and how much effort to put forth according to their own schedule.

The community developers contribute to the development process by reporting bugs and patches. A large number of bug reports and patch reports generally show that the project attracts a lot of attention from the developer community. The core developers are responsible for attracting and motivating the developers by updating the project and releasing the new versions of the project frequently, or by broadcasting the news about the project in the community. The frequent updates attract more developers to participate because their contributions can be valued, and indicate that core developers make effort to promote the project motivate developers in the community. New versions of the project also provide the developers with the new tasks.
C. APACHE

Apache Web server was not initiated by a single developer like other open source software such as Linux Kernel. It was started by the group of people that shared common interests and got to know each other by exchanging information, fixes and suggestions. As the development started, more and more people get attracted and started to help out by sending little patches, suggestions, or replying to mail list, etc. Later on more important contributions were done by the people.

The Apache data sources are developer email list (EMAIL), concurrent version control achieve (CVS) and problem reporting database (BUGDB). Email contain different sort of messages, CVS corresponds to one modification request (MR) and BUGDB contains the problem report number, affected modules, status, name of submitter and date.

Apache development role hierarchy

The role hierarchy of Apache is linear and meritocratic as shown in figure 3. The individuals start contributing in the development of apache web server as end users and advances in their career up to ASF board of directors. When the group felt that the person has earned the merit to be part of the development community, direct access to the code repository is granted. It increases the group and ability of the group to develop the program, and to maintain and develop it more effectively. The membership in Apache is by invitation only. The various roles sets are as follows:

- **End user** - Apache role hierarchy starts with end users. A user is someone that uses the software. Users contribute in the development of Apache projects by providing the feedback to developers in the form of bug reports and feature suggestions. Users participate in the Apache community by helping other users on mailing lists and user support forums.

- **Developer** - The developers are also known as contributors. The developers submit the patches, defect reports, project documentation, suggestions, and criticism, and also participate on developer mailing list, and discussions.

- **Committer** - The developers who have the write access to the code repository and have signed the contributor license agreement (CLA) on a file are called as committers. They have apache.org mail address and make short-term decisions. Committer review the patches and either accept or reject into the source tree. The main responsibilities of committer are as follows:
  - **Deciding release plans and responsibilities** - The major responsibility of the committers is to decide when the branch of the code is ready for release. The releases can not be taken lightly and each release must uphold the Apache tradition of quality. Each Project Management Committee formally authorizes the distribution of releases to the public.
  - **Applying patches** - The committers review, discuss and apply the patches submitted by the volunteers in order to maintain healthy communities. The committers are also responsible for the quality of the code.
  - **Helping user** - Committers help both the developers and users on the projects they work by monitoring them and provide useful responses to questions from users.
  - **Monitoring commits and issues** - Committers are responsible for monitoring Bugzilla/ Jira for bugs, enhancement requests and for reviewing the commit email messages for their projects.
  - **Maintenance of web sites** - The committers are responsible for the maintenance of web sites as the Apache web site and the project web sites need constant maintenance. The Committers on a project are expected to collectively maintain the project's web site. The Apache Committers as a whole share the responsibility to maintain the main Apache site.

![Figure3: Role hierarchy of Apache](image)
• **PMC Member** - The Project Management Community (PMC) is established by the resolution of the board and is responsible for the management of each project within the Apache community. Each PMC consists of at least one officer of the ASF, who shall be designated chairperson, and may include one or more other members of the ASF. The PMC is charged with the responsibility for a top level project and the main role is not to code but is to ensure that all legal issues are addressed, that procedure is followed, and that each and every release is the product of the community as a whole. The PMC is the vehicle through which decision making power and responsibility for oversight is devolved to developers. The long term development of community and to ensure that balances and wide scale peer review and collaboration takes place is also the responsibility of PMC. The board has the faculty to terminate a PMC at any time by resolution.

• **ASF Member** - An ASF member has the right to stand as a candidate for the board election, to elect the board, to propose a committer for membership and to propose a new project for incubation. ASF members are nominated and elected by current ASF members due to merit for the evolution and progress of the foundation. The activities of the members are coordinated through their mailing lists and through annual meetings.

• **PMC Chair** - The chair of the PMC is appointed by the board and is an officer of the ASF. The day-to-day infrastructure for each project is organized by the chair of PMC. The chair has the responsibility to the board and is ultimately responsible for the project thereafter. The PMC and The position of chair is defined by the ASF Bylaws. The Apache project is controlled and lead by the PMC and the chair is the interface between the Board and the project. The various duties of PMC chair are as follows:

  **Report to the board** - The PMC chair report quarterly to the board regarding the status of project and other community legal issues.

  **New PMC member addition** - When the voting is done successfully by the PMC, an email is sent to the board by the chair for taking the acknowledgement. The PMC chair then waits for 72 hours before adding a new PMC member.

  **Change of PMC chair** - If the chair is being changed, then at some stage your PMC needs to send the board an official resolution for the board to approve or reject before this change can officially take place.

• **ASF Board member** - The ASF board is responsible for the management of the corporate assets which includes funds, intellectual property, trademarks, and support equipment and affairs of the corporation in accordance with foundation bylaws. However, technical decision-making authority regarding the content and direction of the Apache projects is assigned to each respective project management committee. Currently the board has nine members, elected between the members of the foundation every year.

Apache board of directors play a very important role in the design, implementation and monitoring of corporate governance and allocate resources to projects. The board is elected annually by the ASF membership. The corporate governance and nominating committee provide assistance to the board of directors for fulfilling its responsibilities by identifying individuals qualified to become directors.

- **APACHE Role Migration:** - The role migration of Apache web server is as shown in figure 4. The developers submit the patches, defect reports, project documentation and participate on the developer mailing lists. These patches are reviewed by the committers and are either rejected or accepted into the source tree.

![Figure 4: APACHE Role Migration](attachment:image.png)
When the project management committee is satisfied with the developer’s contribution, they may elect to extend an offer of committership to the developer, granting him/her write access to the source tree. The developers submit a contributor license agreement (CLA) in order to accept the committership.

Apache software foundation (ASF) is responsible for granting the PMC membership. The developers or the committer must be nominated by an existing ASF member and accepted by a majority vote of the ASF membership participating in the election in order to become a PMC member. The developers or the committers who have demonstrated commitment to the project, good judgment in their contributions to the source tree, and capability in collaborating with other developers are nominated to become the PMC members.

The project management (PMC) is charged with the responsibility for a top level project and is responsible for the management of the project within Apache community. The ASF members are nominated and elected by the majority vote of existing ASF members in the same way as the PMC members are elected. The chair of PMC is also an ASF member who is elected by his/her fellow ASF members. The ASF members may run for office on the ASF board of directors, as outlined by the ASF bylaws. Accordingly, the chairman, vice chairman, president, vice president, treasurer, and secretary are elected annually. The processes associated with the committee are ad hoc and monolithic.

D. MOZILLA

Mozilla is hybrid open source software that has incorporated the quality of commercial development models and the scalability of the FLOSS development models. As opposed to other OSS, Mozilla has a large core development group, and enforces strict code ownership which means code commitment can not be done without approval from a source of authority. OSS projects also rely on decision making mechanisms like voting to manage the chaos of open source development. The incorporation of enhancements and new features in Mozilla also utilizes the voting mechanisms, but the voting process is less formal [16][23]. Mozilla is divided into directories /layout, /mailnews, and so on. Several directories that correspond to modules in Mozilla and are similar in size to Apache project are selected.

- /js contains code for tokenizing, parsing, interpreting, and executing JavaScript scripts.
- /layout contains code for the layout engine that decides how to divide up the “window real estate” among all the pieces of content.
- /editor contains code used for the HTML editor (i.e., Composer in Mozilla Classic), for plain-text and HTML mail composition and for text fields and text areas throughout the product.
- /intl contains code for supporting localization.
- /rdf contains code for accessing various data and organizing their relationships according to Resource Description Framework (RDF), which is an open standard.
- /netwerk contains code for low-level access to the network (using sockets and file and memory caches) as well as higher-level access (using various protocols such as http, ftp, gopher, and castanet).
- /xpinstall contains the code for implementing the SmartUpdate feature from Mozilla Classic [11].

![Figure 5: Role Hierarchy and conceptual model of Mozilla.](image-url)
MOZILLA Role Hierarchy

Mozilla community include all those who are responsible for writing code, developing web pages and applications, testing software, writing documentation and doing any of the multitude of other things that make Mozilla useful and successful. Mozilla.org staff provides the overall guidance for the project which includes building of Mozilla community development of Mozilla, assisting potential new developers, and creating overall policies and procedures for the project [30]. Mozilla role hierarchy consists of developers, module peer, module owner, reviewer, super reviewer, Bugzilla component owner and smoke test coordinators. The roles are as presented in figure 5.

The patch evolution process in Mozilla is shaped around problem statement which may be either the bug or a new enhancement or a feature to be incorporated. The solution to this problem is delivered in the form of a patch. The developer, peer and the module owner are the three parties, who work closely to define the problem and candidate solutions, and resolve the problem. Hence, the set of activities taking place in the relationship of the above parties can be described as define, refine, and resolve. The conceptual model of the patch evolution process space in Mozilla is as shown in figure 5.

The Mozilla data sources are concurrent version system (CVS) and bugzilla problem tracking system. All problem reports are in bugzilla.

- **Developer**- The developers submit the patches. After the source code is obtained, changes are made, built and tested, the changes are to be reviewed and checked. In order to get the changes reviewed and checked, a file listing all the changes is created. This file is called patch. All the changes to the Mozilla source code are presented in the patches [31].

  Patches are submitted by the developers to the Bugzilla by attaching to the appropriate bug report. These patches may serve as a bug fixes or patch that provides enhancement or a new feature. After the patch is submitted, it is reviewed by the reviewer who provides the feedback based on the modifications needed. The developer then enhances the patch in order to make necessary enhancements and resubmit the patch. Once the patch gets accepted, developer commits it to the source code repository if he has the privilege to the source code repository.

  Therefore the developers help Mozilla by fixing bugs, adding new features, writing tests, updating documentation and making Mozilla development easier for others [32].

- **Module peer, Module owner, Reviewer** - Peer developers help the developers in understanding the problem and solution space by investing huge effort in engaging in discussions with the assigned developers before and after the solution development. They do not develop the alternative solutions, and provide the explanation of the problem, identify related problems, propose alternative solutions, verify the developed solution, and spot misalignment between the developed solution and the original problem.

  Peer developer advises the developers when they are not familiar with the problem, or do not have sufficient information to provide a solution. The peer developers also express their opinions on the developer or reviewer comments while discussion around the review, which might affect the new patch that developer is going to submit.

  Code review is a basic mechanism for validating the design and implementation of patches and helps in maintaining a level of consistency in design and implementation practices. Mostly, two levels of review are performed, known as ‘review’ and ‘super-review’. The review ensures that the patch adheres to the initial requirements, commonly accepted standards, does not introduce inadvertent errors and unwanted side effects to the common code base [33].

  Mozilla project is very large and is divided into modules. Each of the modules has a module owner and zero or more peers. Module owner is a person to whom the leadership of a module’s work has been delegated. The module may be a code module or a non-code module. The code module consists of the piece of functionality whereas the non-code modules consist of an activity with well defined boundaries. In code modules, the module owner is responsible for improving code quality, implementing revisions and innovations as appropriate, coordinating development with that of the rest of the codebase, documenting as much as possible, responding appropriately to code contributions, and creating an environment where competent newcomers are welcomed and included [34].

  The patch review is done by the module owners and the peer developers. By default, the review request is forwarded to the module owner. The module owner may assign one of his peers to perform the review. On an average, the peer developers review the patches before the module owners. The peer review helps the developers in finding their mistakes and misalignment with the community standards. Peer developers also contribute before the patch submission by helping the developers better define the problem/solution. Although they do not review the patches as much as the module owners do, but their effect is still valuable in identifying and eliminating immature patches. Only 33.4% of the patches are reviewed by the module owners, 43.8% patches are not reviewed by module owners and 82.8% of the patches are reviewed in 48 hours. It is the responsibility of the developer to contact the module owner and remind him/her to perform the review if there is no response from the reviewer.
The patch review is an iterative process in which the reviewer asks for the modifications to the developer and the developer make the necessary enhancements and resubmit the patch. The reviewer does not fix the code themselves. The resubmission of the patch happens when the patch submitted by the developer is rejected or is needed to be improved. The approval or the rejection of the patch is indicated by flagging the patch review+, super-review+, and ui-review or review-, super-review-, and ui-review- respectively. The review+ flag from each module owner is received for the patch that changes code in more than one module. If the first reviewer feels that the patch would benefit from additional reviews, they should request a second review from an appropriate person. The module owners generally wait for the peer developers to firstly review the patch. If the peer comment result in resubmission of the patch then that saves them one review.

Peer developers and module owners pay high attention to implementation and design aspects of the developed patches. Peer developers provide more functionality and usability feedback to patch developers than module owners, and comment less on documentation and coding style aspects of the developed patches. Module owners mainly focus on quality and long-term maintainability of the project. They do rarely participate in the development. They have a group of peers that do most of the development and are the part of the core development team.

Reviewers are the expensive resources of the community, any reduction in reviews that does not lead to a reduction in the number of defects found will result in cost saving [16].

- **Super Reviewer** - The super reviewers are a designated group of strong hackers who review code for its effects on the overall state of the tree, use of interfaces, overall quality and adherence to Mozilla coding guidelines. The code review which is done by the reviewers is then followed by the super reviewers.

- **Smoke Test Coordinator** - The review process is done by the module peer and module owner in Mozilla. The role of reviewer and smoke test coordinator has been almost omitted from the development process of Mozilla.

![Figure 6: Mozilla Role Migration Process][23]
Reviewers are specific to given areas of code base, but the patch may be reviewed by any super-reviewer. In other words, the reviewer possesses domain expertise in the area the patch addresses where as super-reviewer do not need to possess domain expertise in the area the patch addresses. The super-reviewer may also have domain expertise. If the super-reviewer has domain expertise, how complex the review is likely to be can be easily estimated, super-review can be completed more quickly, and more comprehensive evaluation of the patch can be provided. Super-review focuses on different range of issues than does a code review. To check code into Mozilla, the super-reviewer approval is required, and the super review includes how a patch fits into the broader Mozilla codebase which includes use of APIs, Use of XPCOM, cross module effects and Mozilla coding practices [30][16][34].

**BUGZULLA COMPONENT OWNER** - A Bugzilla component owner is the default recipient of bugs filed against that component. When the bug is filed by someone, the community member to whom that bug will be assigned may be specified by the bug reporter. If bug reporter does not make any assignment, the bug will go to Bugzilla component owner. Bugzilla component owner regularly review bug reports, reassign bugs to correct owners, ensure test cases exist, track the progress toward resolving important fixes, and otherwise manage the bugs in the component. Sometimes, the Bugzilla component owner and the related module owner may be the same person. But in many cases they will be different as their roles are different [33].

**MOZILLA ROLE MIGRATION**
The reviewer reviews the patches and focus on quality and long-term maintainability of the project. The performance of the reviewer is demonstrated by assessing the quality and the effect of submitted patches and the enhancements. If a reviewer believes that s/he has done his job appropriately, the reviewer makes the request for super reviewership. When the request is considered, the reviewer’s performance is accessed and is recommended for super reviewership. The super reviewer will propose the candidate to the remainder of the super reviewers, and discuss the candidate’s performance. The positive and the negative consensus are discussed by the super-reviewer’s. Upon the group consensus, the higher rank is bestowed on the reviewer. In other words, super reviewership is granted on the basis of positive consensus. This process is presented in figure 6.

**APACHE AND MOZILLA DEVELOPMENT COMMUNITY SIZE**
In apache, almost 400 people have contributed the code which was incorporated into a small product. The changes made to problem report are referred as fixes and the changes that added new functionality are referred as code submissions. 182 people have contributed to 695 fixes and 249 people have contributed to 6092 code submissions.

<table>
<thead>
<tr>
<th>PROJECT</th>
<th>DEVELOPERS</th>
<th>PEOPLE CONTRIBUTED TO CODE SUBMISSIONS</th>
<th>PEOPLE CONTRIBUTED TO FIXES</th>
</tr>
</thead>
<tbody>
<tr>
<td>APACHE</td>
<td>388</td>
<td>249</td>
<td>182</td>
</tr>
<tr>
<td>MOZILLA</td>
<td>486</td>
<td>486</td>
<td>412</td>
</tr>
</tbody>
</table>

Table 1: Developers in Apache & Mozilla

In Mozilla, almost 486 people have contributed the code and 412 people have contributed to fixes which has been incorporated as examined from the comment records in CVS. The number of developers in Apache and Mozilla is presented in table 1 and the number of developers in Apache and various Mozilla modules are as presented in table 2.

<table>
<thead>
<tr>
<th>S. No</th>
<th>PROJECT</th>
<th>DEVELOPERS</th>
<th>PEOPLE CONTRIBUTED TO CODE SUBMISSIONS</th>
<th>PEOPLE CONTRIBUTED TO FIXES</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Apache</td>
<td>388</td>
<td>249</td>
<td>182</td>
</tr>
<tr>
<td>2</td>
<td>/layout</td>
<td>174</td>
<td>174</td>
<td>129</td>
</tr>
<tr>
<td>3</td>
<td>/js</td>
<td>127</td>
<td>127</td>
<td>51</td>
</tr>
<tr>
<td>4</td>
<td>/rdf</td>
<td>123</td>
<td>123</td>
<td>79</td>
</tr>
<tr>
<td>5</td>
<td>/network</td>
<td>106</td>
<td>106</td>
<td>74</td>
</tr>
<tr>
<td>6</td>
<td>/editor</td>
<td>118</td>
<td>118</td>
<td>85</td>
</tr>
<tr>
<td>7</td>
<td>/intl</td>
<td>87</td>
<td>87</td>
<td>47</td>
</tr>
<tr>
<td>8</td>
<td>/xpinstall</td>
<td>102</td>
<td>102</td>
<td>64</td>
</tr>
</tbody>
</table>

Table2: Developers in Apache and Mozilla Modules
WORK DISTRIBUTION IN APACHE AND MOZILLA

In Apache, most of the contributions are done by the core developers. Top 15 developers of the total, belong to core developers and the rest belong to non-core developers. The core developers (15 developers) have contributed 83% of the MRs, 83% of the deltas (an MR generates one delta for each of the files it changes), 91% of deleted lines and 88% of added lines. Non-core developers spent very little efforts in the total contributions. In total, 6000 MRs, 18000 deltas and 2,20,000 lines of code are added by the Apache developers.

In Mozilla, there are 22-36 top developers which are considered as core developers. The top developers have contributed 83% of the MRs and 88% of lines added. The work distribution is as presented in table 3.

PROBLEM REPORTS IN APACHE AND MOZILLA

In Apache, the problem reports come from a wider group of participants as examined from BUGDB. Problems are reported by the build, test and customer support teams. There are 3060 people who have submitted 3975 problem reports. The 458 people have submitted 591 reports that have caused the change to the Apache code, and 2654 have submitted 3384 reports that does not trace to code change because the defect was already fixed or the issue was related to incorrect configuration of the product or they did not contain the sufficient detail to reproduce the defect.

Table 3: Work distribution in Apache and Mozilla

<table>
<thead>
<tr>
<th>S. NO</th>
<th>PROJECT</th>
<th>TOP DEVELOPERS</th>
<th>%TOP DEVELOPERS</th>
<th>DELTA(K)</th>
<th>MRs(K)</th>
<th>LINES ADDED (K)</th>
<th>YEARS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Apache</td>
<td>15</td>
<td>4.6</td>
<td>18</td>
<td>6</td>
<td>220</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>/layout</td>
<td>35</td>
<td>20.1</td>
<td>42</td>
<td>12.7</td>
<td>800</td>
<td>2.6</td>
</tr>
<tr>
<td>3</td>
<td>/js</td>
<td>24</td>
<td>18.9</td>
<td>14</td>
<td>4.6</td>
<td>308</td>
<td>2.6</td>
</tr>
<tr>
<td>4</td>
<td>/rdf</td>
<td>26</td>
<td>21.1</td>
<td>12</td>
<td>4.1</td>
<td>274</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>/network</td>
<td>24</td>
<td>22.6</td>
<td>10</td>
<td>3.2</td>
<td>221</td>
<td>1.6</td>
</tr>
<tr>
<td>6</td>
<td>/editor</td>
<td>25</td>
<td>21.2</td>
<td>8</td>
<td>2.9</td>
<td>203</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>/intl</td>
<td>22</td>
<td>25.2</td>
<td>5</td>
<td>2</td>
<td>118</td>
<td>1.8</td>
</tr>
<tr>
<td>8</td>
<td>/xpinstall</td>
<td>22</td>
<td>21.5</td>
<td>5</td>
<td>1.9</td>
<td>113</td>
<td>1.7</td>
</tr>
</tbody>
</table>

In Mozilla, 6837 people have reported 58,000 problem reports. The 1043 people have reported 11,616 problem reports that trace to code changes in Mozilla, and 5794 have submitted 46384 problem reports that do not trace to code changes. The problem report contribution in Apache and Mozilla is as shown in table 4.

The number of problem reports in Mozilla modules is as presented in table 5. The problem report contribution was uniform in Apache, but the contribution varies substantially in Mozilla. The 15 top core developers have produced only 66% of the fixes where as in Mozilla, top reporters have 1000 PRs. The 50% of the PRs are reported by just 113 people, and 46 of these 113 have not contributed any code. In Apache, 15 top reporters have submitted only 213 PRs, 2600 developers submitted one report, 306 have submitted 2 reports, 85 people have submitted 3 reports and maximum of PRs was submitted by one person was 32. There are only 3 core developers in top

Table 4: Problem reports in Apache and Mozilla

<table>
<thead>
<tr>
<th>PROJECT</th>
<th>TOTAL PRs</th>
<th>PEOPLE CONTRIBUTED</th>
<th>PRS CAUSED CHANGE</th>
<th>PRS WITHOUT CHANGE</th>
<th>PEOPLE CAUSED CHANGE</th>
<th>PEOPLE WITHOUT CHANGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache</td>
<td>58000</td>
<td>3060</td>
<td>591</td>
<td>3384</td>
<td>458</td>
<td>2654</td>
</tr>
<tr>
<td>Mozilla</td>
<td>3975</td>
<td>6837</td>
<td>11616</td>
<td>46384</td>
<td>1043</td>
<td>5794</td>
</tr>
</tbody>
</table>

In Mozilla, strict code ownership is enforced. The code must be checked by the module owner before being committed. The module owner is responsible for fielding bug reports, enhancement requests, and patch submissions in order to facilitate good development.
VIII COMPARITIVE CASE ANALYSIS

The Apache coordination approach works well for a small project. The server is kept small and any functionality beyond the basic server can be added by the means of ancillary projects that interact with Apache only through Apache’s well defined interface. The control over this interface is asymmetric. The coordination approach of apache is limited by the stable asymmetric controlled interface and is handled by a small core team using implicit mechanisms.

As there is no strict code ownership, many people can contribute code simultaneously without waiting for approvals and permissions. But in Mozilla, strict code ownership is enforced and the code must be reviewed by the module owner before being committed. Therefore, Apache includes the benefits of speed, productivity and quality.

There is only one path of advancement in Apache. Other less formal communities are conference community, security community and java community process. In Mozilla, there are several ways for potential developers and non-technical people to get involved in the community. The path of advancements includes development, quality assurance, source build, and community management.

The task of finding and reporting the bugs in Apache do not involve the task of changing the code. In other words, these are completely free from interdependencies. The bug fixing is sometimes entangled in interdependencies, but do not cause coordination problems. The Mozilla has also some benefits similar to Apache, but do not include all of the benefits. The significant effort of the community has resulted in low interdependency in bug finding and fixing tasks. The Mozilla modules are not interdependent from one another.

In Apache, there is free style of communication and implicit coordination. As Mozilla modules are too large for a team of 10 to 15 developers to do 80 percent of the work in desired time, free style of communication and implicit coordination is not possible.

The organizational structure of Apache and Mozilla is well defined. OSS projects also rely on decision making mechanisms like voting to manage the chaos of open source development. The voting process that precedes advancement in Apache is formal whereas the voting mechanism of Mozilla is less formal.

Mozilla has large core development team as opposed to Apache. The larger teams have more formal means of coordinating their work and have a module owner with the responsibility of committing the code. This produces the high productivity but at long development intervals.

IX CONCLUSION

Open source software has become famous in the field of software development and millions of people are getting the benefit from this type of development. In this paper, the benefits as well as the challenges of open source software is investigated and is found that in spite of several benefits, open source software is not without several limitations. This paper makes an effort to explore some concepts related to free software, open source software and open source software community. Open source software community is a combination of user group and developer group. The Project leaders guide the vision and direction of a project. The peripheral developers irregularly where as the central developers regularly fix bugs, add features, submit patches, provide support, write documents and exchange other information. Core developers manage CVS releases and coordinate peripheral developers and central developers. Passive users download the code and use it for their needs. Active users discover and report bugs, and suggest new features.

Social or organizational processes affect the performance of software development processes. The Main goal of the comparative study is to examine these processes within two large open source software development projects namely Apache and Mozilla. The different role sets and role migration of Apache and Mozilla in which the developers migrate from peripheral roles to core leadership positions has been identified. The bugs are reported by the developers; peers provide ideas.

<table>
<thead>
<tr>
<th>S.No</th>
<th>MOZZILLA MODULES</th>
<th>PEOPLE REPORTED PRs</th>
<th>PEOPLE WHOSE FIXES ADDED TO CODE</th>
<th>PEOPLE WHOSE BUGS ADDED TO CODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>/layout</td>
<td>3035</td>
<td>129</td>
<td>623</td>
</tr>
<tr>
<td>2</td>
<td>/ls</td>
<td>716</td>
<td>51</td>
<td>147</td>
</tr>
<tr>
<td>3</td>
<td>/rdf</td>
<td>955</td>
<td>79</td>
<td>196</td>
</tr>
<tr>
<td>4</td>
<td>/network</td>
<td>1228</td>
<td>74</td>
<td>252</td>
</tr>
<tr>
<td>5</td>
<td>/editor</td>
<td>857</td>
<td>85</td>
<td>176</td>
</tr>
<tr>
<td>6</td>
<td>/intl</td>
<td>579</td>
<td>47</td>
<td>119</td>
</tr>
<tr>
<td>7</td>
<td>/xpinstall</td>
<td>687</td>
<td>64</td>
<td>141</td>
</tr>
</tbody>
</table>

Table 5: Problem reports in Mozilla modules.
before a patch is developed, review the patches before module owners and find and report errors to developers; and module owners are responsible for the long-term maintainability and usability of Mozilla.

Lastly, through the survey of empirical case studies of the Apache and Mozilla, the results related to the development community size, work distribution, problem reports and the code ownership in Apache and Mozilla are reported. There is no code ownership in Apache and strict code ownership is there in Mozilla.

REFERENCES


WEB REFERENCES


