

بِسْمِ اللَّهِ الرَّحْمَنِ الرَّحِيمِ

**Sudan University of Science and Technology
College of Graduate Studies**



***A Framework for Free and Open Source
Software Best Practices***

إطار عمل لأفضل الممارسات للبرمجيات الحرة والمفتوحة
المصدر

**A thesis submitted in partial fulfillment of the
requirements for the degree of M.Sc in Computer Science**

Prepared by:

Mohammed Sharief Abd El Rahman

Supervisor:

Dr. Mohammed Awad Elshakh

January 2010

Chapter One

Introduction

1.1 Background

In the last ten years, some free software and open source projects have been extremely successful which has attracted the attention of not only the practitioner, but also the business and the research communities. These successes present an important question “what kind of features and innovations offered by these?”

There are many supporters for this direction whose encourage to follow these practices as a new approach for the software development process but still there are many problems not addressed yet, Especially the ad-hoc nature of open source development may result in poor quality software or failures for a number of volunteer projects [1].

In short, OSS is a software whose source code may be freely modified and redistributed with few restrictions, and which is produced by loosely organized team, ad-hoc communities consisting of contributors from all over the world who seldom if ever meet face-to-face, and who share a strong sense of commitment[1].

The basic principle for the OSS Development Process (OSSDP) is that by sharing source code, developers cooperate under a model of systematic peer-review, and take advantage of parallel debugging that leads to innovation and rapid advancement [2].

1.2 Problem Definition

There are many claims associated with open source software which are either misleading or simply false this makes it difficult to really appreciate and exploit the potential of open source software.

However the development processes of these software which are clearly a decentralized approach have many features which could benefit the entire software development processes [3], Furthermore it is appear that the decentralized approach have been followed as a template but each FOSS project apply it is own specific activities and phases. We need to propose FOSS framework that summarized the best practices, activities , contributors and phases of FOSS projects based on some successfully OSS projects taking up some considerations, in order to gain the potential advantages such as productivity (rapid advancement on testing process and bugs fixing), cheapest software, high quality software, innovation and dissemination of knowledge around the world.

1.3 Research Objectives

The objectives of research include:

1. To review activities and phases of OSS projects development and to summarize the best practices. Then to show how OSS project management possible without regularly employed and scheduled software development staff, or without an explicit regime for software engineering project management, why software developers will participate in OSSD projects?
2. To Design and propose a framework for FOSS Development that capture the best practices derived from the first objective.
3. To validate the proposed FOSS framework with one of the existing OSS framework.

1.4 Importance of the Research

The significance of this research dealt with open source software as one of the modern software trends, trying to take advantages of

patterns used to develop this type of programs as one of the methods that can be applied in software development process, in terms of sharing source code, developers cooperate under a model of systematic peer-review, parallel debugging that leads to innovation and rapid advancement.

1.5 Research Limitations

The limitations of this research will be for some designing a management FOSS framework based on the best practices applied in successful OSS project such as Linux, Apache, etc. It aims to identify what to do with little bit explanation telling how to do, working as a boundary that could be used to develop FOSS project within.

1.6 Research Methodology

The research methodology follows the descriptive approach by surveying presentation of OSS project nature, OSS development processes and practices based on some findings extracted from some previous studies that described these projects and provide a comparison between the suggested FOSS framework and Mozilla framework in order to validate it.

1.7 Research Organization

The research has six chapters; the first one is the introduction that gives background about the problem, objective of the thesis and its scope. The literature review is divided in two chapters free and open source software projects nature and free and open source software development processes. Chapter four discusses the traditional software life cycle models and proposes a framework to

develop FOSS projects. Chapter five presents a case study of Mozilla web browser as a successful OSS project with attention to catch the similarities between Mozilla framework and the suggested FOSS framework. Chapter six denotes conclusions, recommendations and future work.

Chapter Two

Free and Open Source Software Projects Nature

2.1 Introduction

It is frequent to make a distinction between the terms ‘free software’ and ‘open source software’. Free software refers not to price but to liberty to modify and redistribute source code. The Free Software Foundation [4], founded by Richard Stallman, advocates the use of its GNU General Public License (GPL) as a copyright license which creates and promotes freedom. He writes “to understand the concept, you should think of free speech, not free beer” [5]. The term ‘open source’ was coined by a group of people concerned that the term ‘free software’ was anathema to businesses, this was resulted in the creation of the Open Source Initiative (OSI) [6]. We use the acronym FOSS for both movements for the sake of simplicity and because both movements share most of their practical goals and

follow similar development processes. The OSI definition includes the following criteria [7]:

- Free redistribution:

The license shall not restrict any party from selling or giving away the software as a component of an aggregate software distribution; no royalty or fee is required for such sale.

- Source code availability: the program must include source code.
- Ability of derived works: modifications and derived works are allowed, not necessarily subject to the same license as the original work.
- Integrity of author's source code: derived works must carry different names or version numbers than the original work.
- No discrimination against persons or groups or fields of endeavor.
- Distribution of license: no need of any additional license.
- License must not be specific to a product and must not restrict other software.

GNU GPL, BSD, Apache, MPL (Mozilla) and Artistic (Perl) licenses are all examples of licenses that conform OSI definition [7].

Another distinction can be drawn between OSS projects that result from the initiative of a given individual or group of individuals, and OSS projects that are supported by, or organized within, industrial software companies. The consequences are noticeable in the way these projects are managed (e.g. composition of the steering committee, decision-making processes) and through the existence of peripheral processes under the exclusive responsibility of the company which backs the project (mainly quality insurance processes).

Free and Open Source Software (FOSS) refers to software whose licenses give users

Four essential '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 [8].

OSS projects can also be classified into communities of interest, centered about the production of software for different application domains, such as games, Internet infrastructure, software system design, astronomy, etc. This factor has a low impact on how the software is produced [9].

FOSS users do not pay royalties as no copyright exists, in contrast to proprietary software applications which are strictly protected through patents and intellectual property rights [10].

2.2 Free Software Foundation (FSF)

In January 1984 one of the original MIT AI Labs hackers, Richard M. Stallman, quit his job at MIT and founded Free Software Foundation (FSF). He objected to the increasing commercialization of university software research [4]. Stallman feared that despite the fact that popular Unix standards like Sun's were broadly distributed, they still remained under private ownership and would be used for proprietary advantage, which is what happened by the early 1990s.

Stallman's goal was to develop software for anyone to use at no cost, thereby implicitly helping software research. He started by

designing a Unix-compatible operating system, GNU. The name GNU was chosen following a hacker tradition, as a recursive acronym for GNU's Not Unix [10]. He chose to make the system compatible with UNIX so that it would be portable, and so that UNIX users could easily switch to it. But in fact that he did not success as well because the GNU operating system was delayed, and the work started with a C compiler and the editor GNU Emacs. Also, commercial UNIX systems were still expensive, and no one got the source code anyway.

2.3 Open Source Initiative (OSI)

The Open Source Initiative (OSI) is a California public benefit corporation, it is an organization founded in 1998 by Eric S. Raymond to promote open source software and development strategies. The OSI are the stewards of the [Open Source Definition \(OSD\)](#) and the community-recognized body for reviewing and approving licenses as OSD-conformant [4].

Eric Raymond wanted to find a way to promote the free software ideas (essentially because he wasn't happy with Microsoft), but he was also concerned that the alternative Free Software Foundation's strong political anti business message was keeping the world at large from really appreciating the power of free software.

The OSI is actively involved in Open Source community-building and education. [OSI Board](#) members frequently travel the world to attend Open Source conferences and events, meet with open source developers and users, and to discuss with executives from the public and private sectors about how Open Source technologies, licenses, and models of development can provide economic and strategic advantages [7].

2.3.1 FOSSD Investors

There are many corporations started early adopting and investing in such FOSS projects [8]:

- Large corporations/enterprises:
 - IBM-Eclipse, Sun-NetBeans, Sun-OpenOffice, HP-Gelato, Apple-*Darwin*, Microsoft Research-*Rotor*, etc.
 - Barclays Global Investors, DKW, Merrill Lynch, etc.
 - DoD, DoE, NSF, NIH, NASA, etc.
 - MIT, Stanford, CMU, UC, UMichigan, etc.
- Mid-size corporations:
 - RedHat, Novell, Borland
- Small (start-up) companies:
 - ActiveState, Collab.Net, Jabber, Ximian, JBoss, Compiere, etc.

2.3.2 A Successful Story of Linux

Linux project could be seen as a great try to face the commercialization and monopolistic path in the software industry, started by Linus Torvalds, the history of Linux started by the summer of 1990 when Linus Torvalds, a student of technology at the University of Helsingfors in Finland, started hacking on an embryo to an Intel 386 Unix system as a hobby project. After a few months he had successfully written a working kernel [5]. Although there was still much to be done, the project drew the attention of curious programmers when Torvalds announced it to a newsgroup on the Internet.

In October 1991 he released the source code for the first ten people to download Linux, It was the beginning of what became a global hack, involving millions of lines of code contributed by

thousands of programmers [5]. It was the beginning of one of the most spectacular software developments ever seen. Many also regard Linux as the best brand of UNIX. Linux represents the philosophy of UNIX— simplicity, portability, and openness. It is the most widely ported system available today, and the only system gaining market share besides Microsoft Windows NT.

The immediate interest was due to the fact that the entire source code was available for free download to anyone who was interested in using and modifying the system. By releasing the source code for free at a very early stage and also updating the releases often, Torvalds quickly found help, support, and feedback from other programmers. Even as the first official version was released in 1994, changes were being made on a daily and weekly basis while Linux continued to mature into a powerful and versatile operating system.

A small development team did not develop Linux in the traditional way of both commercial software development and freeware like GNU. Linux was developed by a huge number of volunteers coordinated through the Internet by a project leader. Quality was maintained by the extremely simple technique of releasing frequently and getting feedback from many users [4].

2.4 Problems with Traditional Development Software Projects

Proponents of open source argue that ‘traditional’ software development projects suffer from various ills. Such projects have been shown to be prone to time and cost overruns, are largely unmaintainable, with questionable quality and reliability [10].

These failures are ascribed to:

- Inadequate understanding of the size and complexity of IS 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 exacerbated by the price of license fees for software and tools required for application development as well as add-on costs for exchange controls.

2.5 Benefits of FOSS

Does OSS have answers to these problems? It is claimed that OSSD produces reliable, high quality software in less time and with less cost than traditional methods. In addition that OSSD is the “most efficient” way to build applications. OSSD can potentially “change, perhaps dramatically, the way humans work together to solve complex problems in computer programming”.

Eric Raymond published that OSSD follow a “bazaar style” which is a loosely centralized, cooperative community where collaboration and sharing enjoy religion status. Conversely, traditional software engineering seems to follow a “cathedral style” where hierarchical structures exist and little collaboration takes place.

Furthermore if we look at the main features of many FOSS models we find that its:

- Collaborative, parallel development involving source code sharing and reuse.
- Collaborative approach to problem solving through constant feedback and peer review.

- Large pool of globally dispersed, highly talented, motivated professionals.
- Extremely rapid release times.
- Increased user involvement as users are viewed as co-developers [10].

The following part will show that OSS approach add some valued features and cover some areas of defects that may found in the traditional software including:

2.5.1 Quality Software

It is maintained that OSS features result in quality software as collaborative development allows for multiple solutions. At the same time there is little tolerance for failure to adhere to the tacitly accepted norms [11].

2.5.2 Development Speed

Reuse of code implies speedier development: the more people there are creating code and adding value to a project, the quicker the product is released and becomes valuable to a user group [10]

2.5.3 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. If encouraged, users can assist developers in finding system faults and improvements, thereby reducing the need (and cost) for extra developers to perform the same function [10].

2.5.4 Access to Existing Code

Developers have access to the “open source toolset”, a huge amount of open source project code which can improve the developer's skills and experience, so that can speed up development.

2.5.5 Collaboration

A further important feature of the FOSSD model is the nature of the development community. Large numbers of geographically dispersed programmers are joined by the internet to produce complex software; they do so largely without pay.

2.5.6 Cost

Total cost of ownership (TCO) of FOSS is widely debated, a basic tenet of free open source is that all source code is free and available to any user to modify and improve. In some phases of ownership, there is evidence that FOSS may have advantages in the area of TCO. OSS can be tested without cost. OSS has no license fees, removing the necessity to purchase additional licenses as the organization grows [7].

Chapter Three

FOSS Development Processes (FOSSD)

3.1 Introduction

This chapter explores patterns and processes that emerge in free open source software development (FOSSD) projects with results

from recent studies of FOSSD. FOSSD is a relatively new way for building and deploying large software systems on a global basis, and differs in many interesting ways from the principles and practices traditionally advocated for software engineering. Hundreds of FOSS systems are now in use by thousands to millions of end-users, and some of these FOSS systems entail hundreds-of-thousands to millions of lines of source code [8]. So what's going on here, and how are FOSSD processes that are being used to build and sustain these projects different, and how might process modeling and simulation techniques be used to explore what's new or different?.

One of the more significant features of FOSSD is the formation and enactment of complex software development processes performed by loosely coordinated software developers and contributors [12]. These people may volunteer their time and skill to such effort, and may only work at their personal discretion rather than as assigned and scheduled. Further, these developers generally provide their own computing resources, and bring their own software development tools with them. Similarly, FOSS developers work on software projects that do not typically have a corporate owner or management staff to organize, direct, monitor, and improve the software development processes being put into practice on such projects.

There are many kinds of key questions that may be addressed in this section such:

- How are successful FOSSD projects and processes possible without regularly employed and scheduled software development staff, or without an explicit regime for software engineering project management?
- Why will software developers participate in FOSSD projects?
- Why and how are large FOSSD projects sustained?

- How are large FOSSD projects coordinated, controlled or managed without a traditional project management team?

Why and how might these answers to these questions change over time? These are the kinds of questions that will be addressed in this section.

3.2 FOSS Development Processes (FOSSD)

Unlike the software engineering world, FOSS development communities don't seem to readily adopt modern software engineering processes. FOSS communities develop software that's extremely valuable, generally reliable, globally distributed, made available for acquisition at little or no cost, and readily used in its associated community [12].

FOSSD is not "software engineering" [13]:

- Different: FOSSD can be faster, better, and cheaper than SE in some circumstances as we will see later in this section.
- FOSSD teams use 10-50 OSSD tools and communications applications to support their development work while traditional SE team does not need such number of tool and communication support.

The next section will introduce example of some FOSSD practices, furthermore, it appears that these processes occur concurrently, rather than strictly ordered as in a traditional life-cycle model or partially ordered as in a spiral process model seen in the next chapter.

3.2.1 Requirements Analysis and Specification

Software requirements analysis helps to identify the problems that a software system should address and the form solutions might

take. It identifies an initial mapping of problems to system-based solutions [13].

Generally FOSS Requirements/Designs could be described as follow:

- Not explicit.
- Not formal.
- Embedded within “FOSSD informalisms” Example FOSS informalisms to follow (as email lists, project Web site, discussion forums, etc).
- FOSS Requirements/Design processes is different from their SE counterparts.

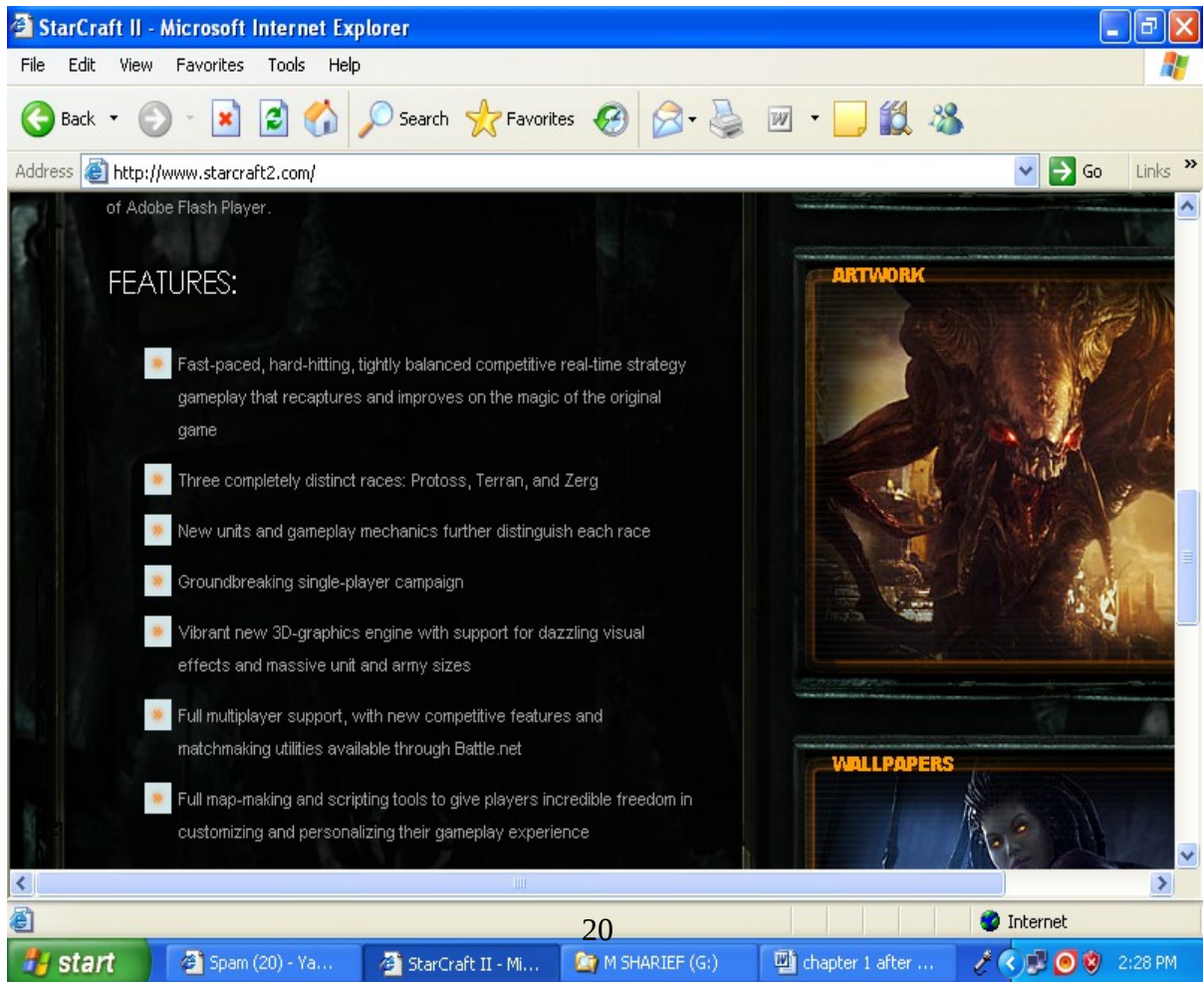
FOSS requirements take the form of threaded messages or discussions on Web sites that are available for open review, elaboration, refutation, or refinement [6].

They routinely emerge as a by-product of community discourse about what its software should or shouldn't do and who'll take responsibility for contributing new or modified system functionality, Figure 3.1 shows the main features of OSS game that have been added and which needed. It appears after assertions in private and public email discussion threads, ad-hoc software artifacts (such as source code fragments included in a message), and site content updates that continually emerge.

More conventionally, requirements analysis, specification, and validation aren't performed as a necessary task that produces a mandated requirements deliverable.

Instead, you find widespread practices that imply reading and sense-making of online content.

In short, requirements take these forms because FOSS developers implement their systems and then assert that certain features are necessary. They don't result from the explicitly stated needs of user representatives, focus groups, or product marketing strategists.



FEATURES:

- Fast-paced, hard-hitting, tightly balanced competitive real-time strategy gameplay that recaptures and improves on the magic of the original game
- Three completely distinct races: Protoss, Terran, and Zerg
- New units and gameplay mechanics further distinguish each race
- Groundbreaking single-player campaign
- Vibrant new 3D-graphics engine with support for dazzling visual effects and massive unit and army sizes
- Full multiplayer support, with new competitive features and matchmaking utilities available through Battle.net
- Full map-making and scripting tools to give players incredible freedom in customizing and personalizing their gameplay experience

ARTWORK



WALLPAPERS



Figure 3.1: Computer game software requirements [14]

This figure explain how developers firstly determine the suggested feature for their software and exchange ideas about the suitable and better requirements after they assert in private and public discussion until reach this final template for the requirements.

3.2.2 Coordinated Version Control, System Build, and Staged Incremental Release

Software version control tools such as the Concurrent Versions System CVS serve as both a centralized mechanism for coordinating FOSS development and a venue for mediating control over which software enhancements, extensions, or upgrades will be checked in (inserted) to the archive [16]. If checked in, these updates will be available to the community as part of the alpha, beta, candidate, or official released versions, as well as the daily-build release, figure 3.2 shows one such a FOSS repository on the Web of a software source code files.

This coordination is necessary because decentralized code contributors and reviewers might independently contribute software updates or reviews that overlap, conflict, or generate unwanted side effects.

Each project team or CVS repository administrator must decide what can be checked in and who can and can't check in new or modified software source code content. Some projects make these policies explicit through a voting scheme [17].

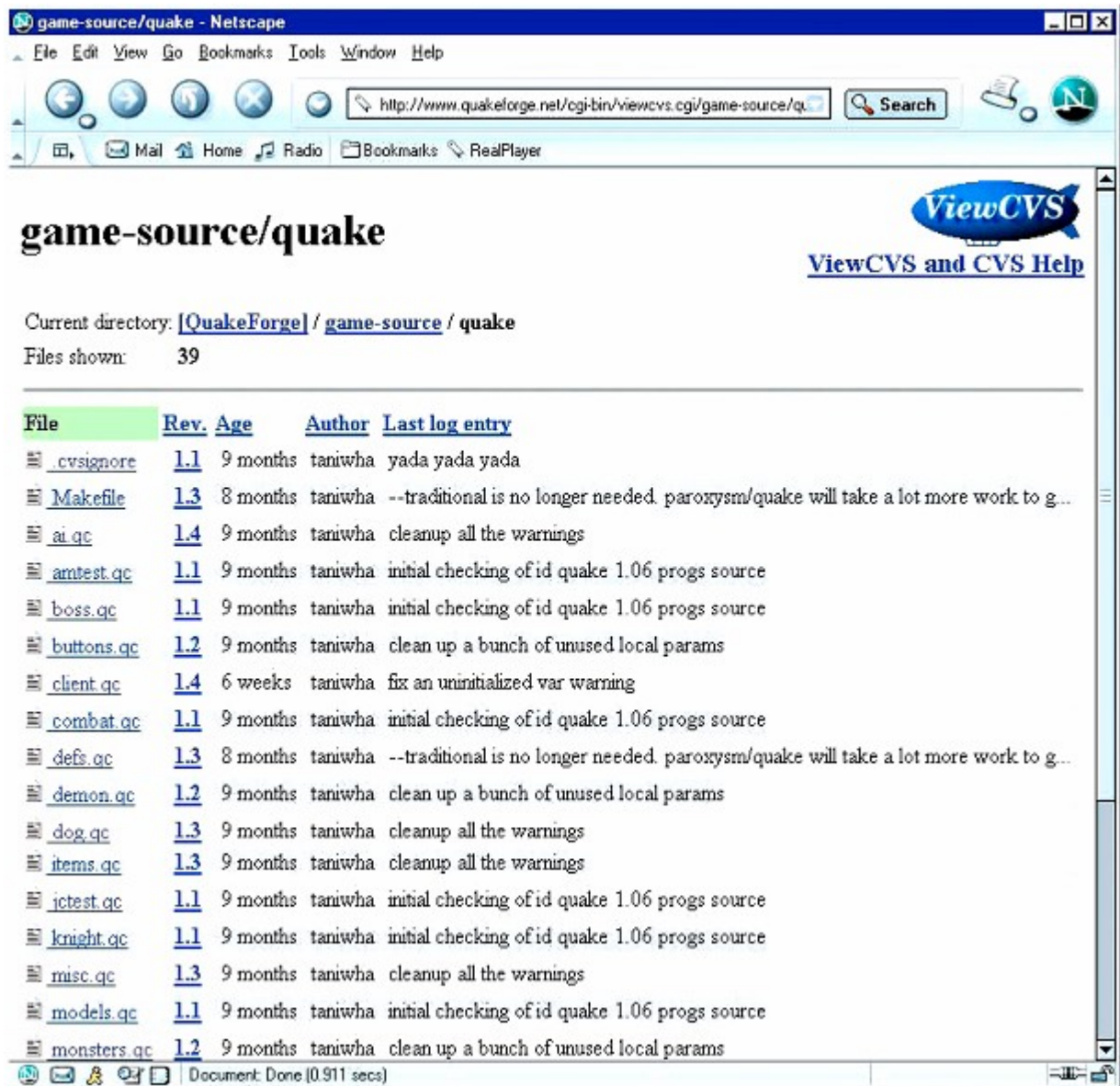


Figure 3.2: A view into a Web-accessible CVS (Concurrent Versions System) configuration archive of software source code files for the game [18].

3.2.3 Maintenance as Evolutionary Redevelopment, Reinvention and Revitalization

In FOSS development community's maintenance is generally viewed as the major activity associated with a software system across its life cycle [13]. However, the traditional label of software maintenance doesn't quite fit what you see occurring in different FOSS communities.

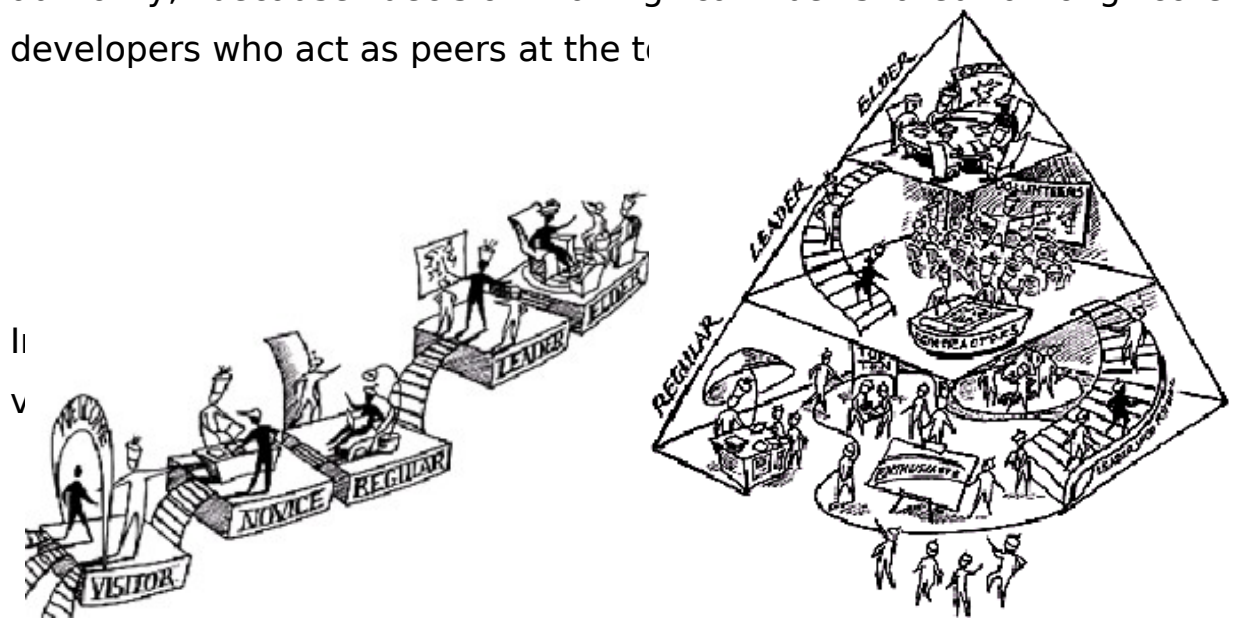
Instead, it might be better to characterize the overall evolutionary dynamic of FOSS as reinvention. Reinvention enables continuous improvement and occurs through sharing, examining, modifying, and redistributing concepts and techniques that have appeared in closed-source systems, research and textbook publications, conferences, and developer-user discourse across multiple FOSS projects.

FOSS systems seem to evolve through minor improvements or mutations that are expressed, recombined, and redistributed across many releases with short life cycles. FOSS end users who act as developers or maintainers continually produce these mutations that appear initially in daily system builds.

3.2.4 Project Management and Career Development

FOSSD projects self-organize as a pyramid via virtual project management, VPM requires people to act in leadership roles based on skill, availability, and belief in project community [13]

FOSS development teams can take the organizational form of interlinked layered operating as a dynamically organized but loosely coupled virtual enterprise. A layered is a hierarchical organizational form that centralizes and concentrates certain kinds of authority, trust, and respect for experience and accomplishment within the team as show in Figure 3.3. However, it doesn't imply a single authority, because decision-making can be shared among core developers who act as peers at the t



and assure the quality of FOSS development activities. It could invite or encourage system contributors to come forward and take a shared, individual responsibility that'll serve to benefit the FOSS collective of user-developers.

VPM requires several people to act as team leader, subsystem manager, or system module owner in either a short- or long-term manner. People take roles on the basis of their skill, accomplishments, availability, and belief in community development. Figure 3.4 shows an example of VPM.

PlaneShift - Netscape

File Edit View Go Bookmarks Tools Window Help

http://www.planeshift.it/main_01.html

Search

Mail Home Radio Bookmarks RealPlayer

PLANESHIFT

News Features Pics

Setting

Player guide

Community

Download

Help Us!

Recruitment
Others

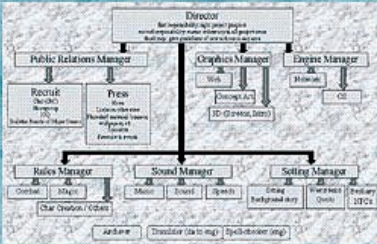
JOIN THE TEAM!

PlaneShift is a complex project and first of all it needs a good organization, for this reason we have divided the project in different departments. Each department has a leader that will ensure the progress and coordination of contributors.

Here you find an Organization Chart that explains which are the departments that you can choose for contributing. Click on it!

In the current state we are not a commercial organization so we can't provide a salary for contributors, members or leaders.

Please note that we accept only people with age of 16 or older. Our team is made of people from 18 to 33 actually.



```

graph TD
    Director[Director] --> PRM[Public Relations Manager]
    Director --> GM[Graphics Manager]
    Director --> EM[Events Manager]
    PRM --> Recruit[Recruit]
    PRM --> Press[Press]
    PRM --> Public[Public Relations Manager]
    GM --> Web[Web]
    GM --> Design[Design Art]
    GM --> ID[ID Graphics Design]
    EM --> Events[Events]
    EM --> CS[CS]
    PRM --> RM[Races Manager]
    PRM --> SM[Sound Manager]
    PRM --> TM[Texture Manager]
    RM --> Create[Character Creation / Races]
    SM --> Audio[Audio]
    SM --> Director[Director (Strategy)]
    SM --> Speech[Speechwriting (ongo)]
    TM --> Create[Character Creation / Races]
    TM --> Director[Director (Strategy)]
    TM --> Speech[Speechwriting (ongo)]
  
```

POSITIONS IN THE TEAM

LEADER:

To be a leader you must pass the approval of the director. Before that you will be considered a W.T.B. (Want To Be) Leader and only after proving that you have the right skills and dedication to the project you will officially become a leader.

There's one leader for each department and he can have also one co-leader helping in his job.
He will ensure progress in his department completing the most important tasks in his area and will organize work of other members.
He is the primary reference for development.
Required Skills:

- ♦ Strong commitment to the project.
- ♦ Good skill to organize work of the Team.
- ♦ Team leadership.
- ♦ Good knowledge of the area in which he applies

The leader is the most important contributor of his department!
He will complete critical tasks, he will always have job to do. His tasks are similar to the ones of the members (see below in the section of a specific department).
He will also manage work of other guys.

Copyright © 2001 PlaneShift Team.
All material in this site under
[PlaneShift License](#)

Document: Done (1.442 secs)

Figure 3.4 a description of how a FOSS computer game development project organizes and manages itself. [21].

3.2.5 Software Technology Transfer

FOSS technology transfer from existing Web sites to organizational practice is a community and project team building process, Not (yet) an engineering process.

It's instead a socio technical process that entails the development of constructive social relationships. Informally it is a negotiated social agreement and a routine willingness to search, browse, download, and try out FOSS assets [13]. Although the Software technology transfer is an important process but it seems often to be a neglected process in the academic software engineering community [19].

FOSS developers publicize and share their project assets by adopting and using FOSS project Web sites as a community wide practice, they build these Web sites using OSS content management systems (such as PHP-Nuke) and serve them using OSS Web servers (Apache), database systems (MySQL). User and developers are increasingly accessing these sites via OSS Web browsers (Mozilla).

FOSS systems, development assets, tools, and project Web sites serve as a venue for socializing, building relationships and trust, sharing, and learning with others. Some open source software projects have made developing such social relationships their primary project goal.

Generally free and open source software development practices give rise to a new view of how complex software systems can be constructed, deployed, and evolved without adhering to traditional software engineering life-cycle principles [13]. Because they rely on electronic communication media, virtual project management, and version management mechanisms to coordinate globally dispersed development efforts. So we can say that these FOSS processes and practices offer new directions for developing complex software systems.

3.3 Results from Recent Studies of OSSD

There are two kinds of studies that offer some insight or findings on OSSD practices each in turn reflects on different kinds of processes which are not well understood at this time. **First**, there are trade studies that focus on convenience surveys of software or IT industry professionals who are early adopters of OSS techniques. **Second**, there are systematic empirical studies of OSSD projects using small/large research samples and analytical methods drawn from different academic disciplines.

3.3.1 Trade/Industry Studies

Among the more widely identified industry studies are those that have been sponsored and published by CIO magazine (www.cio.com), starting back in 2005. These studies of the opinions and experiences of hundreds of IT managers and executives in a variety of enterprise settings report the following kinds of findings:

1. In these enterprises, OSSD projects are primarily targeted to new system deployments, rather than to supporting or replacing existing business system applications.
2. The primary benefits for engaging OSSD projects include anticipation of lower total cost of ownership (TCO), lower capital investment, and greater reliability of the resulting systems.

3. The perceived risks or weaknesses associated with in-house OSSD projects include lack of in-house OSSD skills or OSS developers in the market and uncertainty over the costs of switching from current approaches and vendors to OSS oriented ones.

From the perspective of software process modeling and simulation, the following kinds of observations appear:

1. The costs associated with OSSD projects are unclear, as are the methods for accounting for them and associating them with different OSSD processes or activities.
2. If the surveys participants work in enterprises that explicitly manage their traditional software development processes, they recognize that OSSD projects seem to require different, less familiar processes that may not be well understood by their current software development staff.

3.3.2 Findings from OSSD Research Studies

Rather than attempt to survey the complete universe of studies in these collections, the choice instead is to just briefly sample a small set of studies that raise interesting issues or challenging problems for software process modeling and simulation. Furthermore, it is important to recognize that OSSD is no silver bullet that resolves the software crisis. Instead it is fair to recognize that most of the nearly 100,000 OSSD projects associated with Web portals like SourceForce.org have very small teams of two or less developers [22], and many projects are inactive or have yet to release any operational software. However, there are now at least a few thousand OSSD projects that are viable and ongoing, so that there is a sufficient universe of diverse OSSD projects to investigate, and to model and

simulate their software processes, we will show some of these findings in terms of:

3.3.2.1 Motivating, joining, participating, and contributing to OSSD projects:

One of the most common questions about OSSD projects is why software developers will join and participate in such efforts, often without pay for sustained periods of time. A number of surveys of OSS developers have posed such questions, and the findings reveal the following.

1. OSS developers generally find the greatest benefit from participation is the opportunity to learn and share what they know about software system functionality, design, methods, tools, and practices associated with specific projects or community leaders.
2. OSS developers appear to really enjoy their OSSD work [23], and to be recognized as trustworthy and reputable contributors [24].
3. OSS developers also self-select the technical roles they will take on as part of their participation in a project [25], rather than be assigned to role in a traditionally managed SE project, where the assigned role may not be to their liking.

3.3.2.2 Alliance formation and inter-project social networking:

Gathering of individual OSS developers give rise to a more persistent project team or self-sustaining community. These software developers find and connect with each other through OSSD Web sites and online discourse (e.g., threaded email discussions) [26], and they

find they share many technical competencies, values, and beliefs in common [21,27]. Becoming a central node in a social network of software developers that interconnects multiple OSS projects is also a way to accumulate social capital and recognition from peers.

Thus interesting problems arise when investigating how best to model or simulate the processes of alliance formation and inter-project social networking, and how such processes facilitate or constrain OSSD activities, tool usage, and preference for which development artifacts are most valued by project participants.

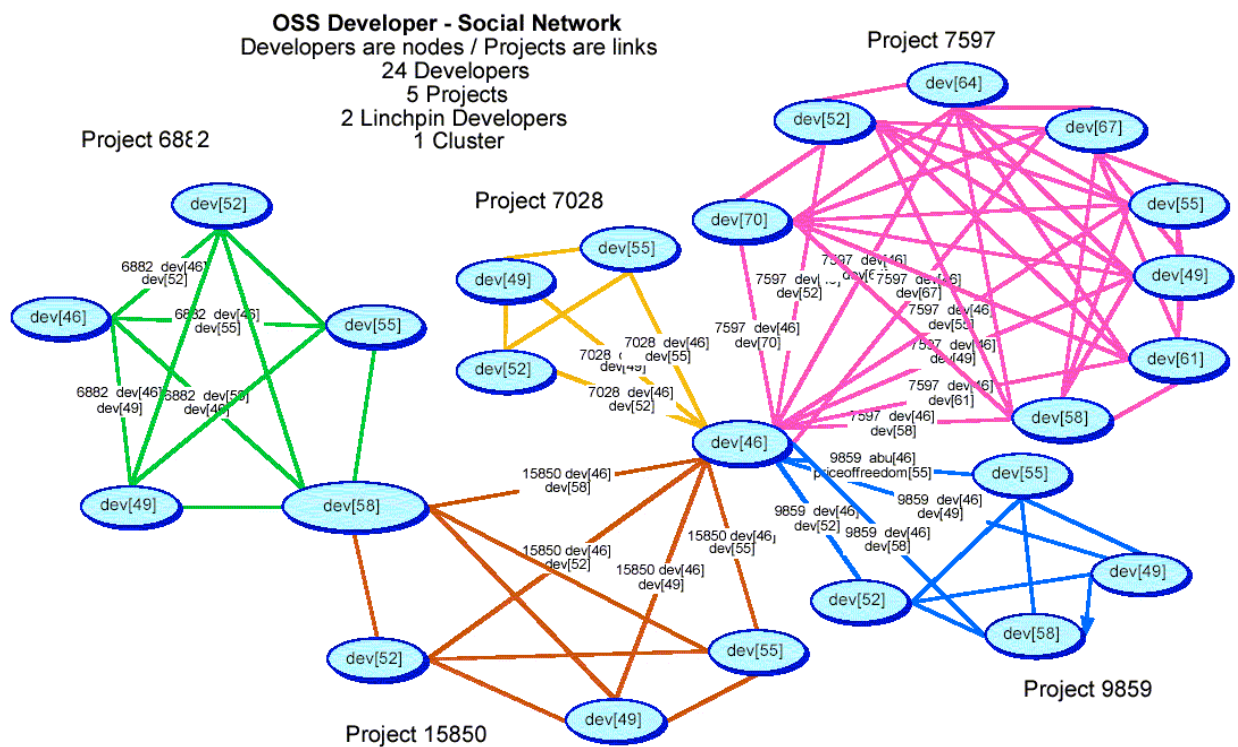


Figure 3.5: a social network that links 24 developers in five projects through two key developers into a larger OSS project community [22]

Chapter Four

Life Cycle Models and the Proposed FOSS Framework

4.1 Introduction

As in any other engineering discipline, software engineering also has some structured models for software development. This chapter was divided into three sections the first will provide a generic overview about different software development methodologies, the second section specified to the existing FOSSD models and the third section is a proposition of a framework for the FOSS software development.

4.2. Software Development Life Cycle (SDLC)

A software life cycle model depicts the significant phases or activities of a software project from conception until the product is retired. It specifies the relationships between project phases,

including transition criteria, feedback mechanisms, milestones, baselines, reviews, and deliverables. Typically, a life cycle model addresses the phases of a software project: requirements phase, design phase, implementation, integration, testing, operations and maintenance

Software life cycle models describe the interrelationships between software development phases. The common life cycle models are:

4.2.1 [Waterfall Model](#)

The least flexible of the life cycle models, also known as Classic Life Cycle Model (or) Linear Sequential Model (or) Waterfall Method. Still it is well suited to projects which have a well defined architecture and established user interface and performance requirements. The waterfall model *does* work for certain problem domains, not ably those where the requirements are not well understood [28].

The standard waterfall model for systems development is an approach that goes through the following steps (activities) [29]:

1. Document System Concept
2. Identify System Requirements and Analyze Them
3. Break the System into Pieces (Architectural Design)
4. Design Each Piece (Detailed Design)
5. Code the System Components and Test Them Individually (Coding, Debugging, and Unit Testing)
6. Integrate the Pieces and Test the System (System Testing)
7. Deploy the System and Operate It

This model is widely used on large government systems; particularly by the Department of Defense (DOD). The standard waterfall model is associated with the failure or cancellation of a number of large systems, it can also be very expensive [29]. As a result, the software

development community has experimented with a number of alternative approaches including [27]:

- Spiral Design (Go through waterfalls, starting with a very rough notion of the system and becoming more detailed over time)
- Modified Waterfalls (Waterfalls with Overlapping Phases; Waterfall with Subprojects)
- Evolutionary Prototyping (Start with initial concept, design and implement an initial prototype, iterate as needed through prototype refinement until acceptable, complete and release the acceptable prototype)
- Staged Delivery (Go through Concept, Requirements Analysis, and Architectural Design - then implement the pieces, showing them to the customer as the components are completed - and go back to the previous steps if needed)
- Evolutionary Delivery (a cross between Evolutionary Prototyping and Staged Delivery)

4.2.1.1 Advantages of Waterfall Model

1. Clear project objectives.
2. Stable project requirements.
3. Progress of system is measurable.
4. Strict sign-off requirements.

4.2.1.2 Disadvantages of Waterfall Model

1. Time consuming
2. Never backward (Traditional) between phases.
3. Little room for iteration.
4. Difficulty responding to changes in the requirements. [45]

4.2.2 Prototyping Model

It was advocated by [Brooks](#) in early 60th, Useful in situations where requirements and user's needs are unclear or poorly specified. The approach is to construct a quick and dirty partial implementation of the system during or before the requirements phase because many aspects of the system are unclear until a working prototype is developed. Typical implementation language is scripting language and UNIX shell (due to availability huge amount of components that can be used for construction of the prototype) [27].

Prototyping consists of developing a partial implementation of the system to give the users a feel for what the developer has in mind. The users then give feedback on what they think of the prototype - what works and what doesn't - and the developer can make changes more easily and efficiently than if the changes were to be made later on in development.

4.2.2.1 Advantages of Prototype Model

1. User interaction available in during development cycle of prototype.
2. Missing functionality can be identified easily.
3. Confusing or difficult functions can be identified.
4. Helps to refine the potential risks associated with the delivery of the system being developed
5. Helps to deliver the product in quality easily.
6. Environment to resolve unclear objectives, various aspects can be tested and quicker feedback can be got from the user.
7. Encourages innovation and flexible designs.

4.2.2.2 Disadvantages of Prototype Model

1. Contract may be awarded without rigorous evaluation of Prototype.
2. Identifying non-functional elements difficult to document.
3. Incomplete application may cause application not to be used as the full system was designed.
4. Incomplete or inadequate problem analysis.
5. Client may be unknowledgeable.
6. Approval process and requirement is not strict, structure of system can be damaged since many changes could be made.
7. Over long periods, can cause loss in consumer interest and subsequent cancellation due to a lack of a market (for commercial products).
8. Not suitable for large applications. [⁴⁶]

4.2.3 Spiral Model

A better model, the "spiral model" was suggested by Boehm in 1985; the spiral model provides useful insights into the life cycle of the system. It could be considered as a generalization of the prototyping model [³⁰]. That why it is usually implemented as a variant of prototyping model with the first iteration being a prototype. But it also supposes unlimited resources for the project, No organization can perform more then a couple iterations during the initial development of the system, the first iteration is usually called prototype as shown in Figure 4.1 Spiral model phases.

The Spiral model of development is risk-oriented; each spiral addresses a set of major risks that have been identified, Figure 4.2 provide detailed view of spiral model phases. Each spiral consists of: determining objectives, alternatives, and constraints, identifying and

resolving risks, evaluating alternatives, developing deliverables, planning the next iteration, and committing to an approach to the next iteration [29].

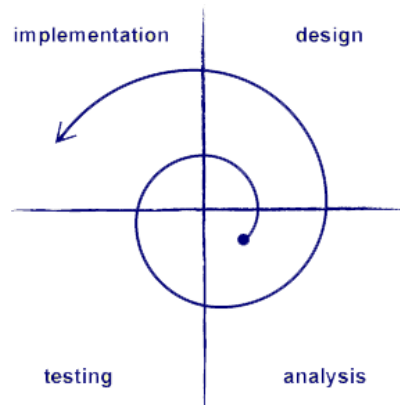


Figure 4.1: Spiral model phases

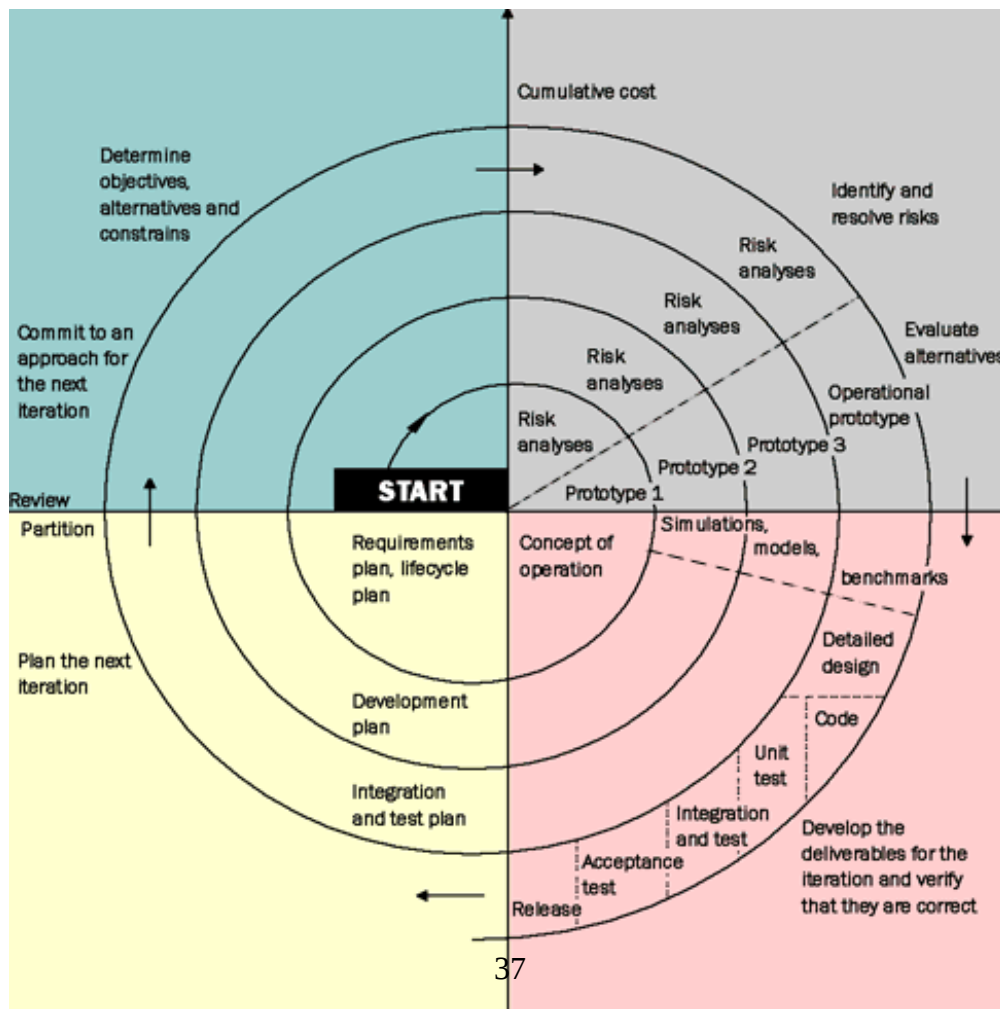


Figure 4.2: Detailed Spiral model phases [29]

4.2.3.1 Advantages of Spiral Model

1. Avoidance of Risk is enhanced.
2. Supports for dynamically changing requirements.
3. Strong approval and documentation control.
4. Implementation has priority over functionality.
5. Additional Functionality can be added at a later date.

4.2.3.2 Disadvantages of Spiral Model

1. Highly customized limiting re-usability.
 2. Applied differently for each application.
 3. Risk of not meeting budget or schedule.
 4. Requires expertise in risk evaluation and reduction
- Complex, relatively difficult to follow strictly. [29]

4.2.4 Evolutionary Prototyping Model

This is kind of mix of Waterfall model and prototyping. Might be suitable in projects where the main problem is user interface requirements, but internal architecture is relatively well established and static. It is Cost effective (Development costs reduced better than others models) and combine both features of Waterfall model and Prototyping [46].

4.3 Recent Software Process Challenges

There are many challenges that facing software development process, however we find that OSS approach cover some of these areas in its development practices such as how to maintain global software development, easily process improvement because OSSD processes often iterate daily versus infrequent singular software life cycle engineering events which could make it easier to improve, but still yet there are no solid solutions to face these challenges such as process improvement, here we will summarize some of these challenges that facing both: Distributed, decentralized, and/or global software development

4.3.1 Process Improvement

Process improvement is a series of actions taken by a Process Owner to identify, analyze and improve existing processes within an organization to meet new goals and objectives [13]. These actions often follow a specific methodology or strategy to create successful results. Samplings of these are listed below:

1. Benchmarking.
2. Business Process Improvement.
3. Business process reengineering.
4. Capability Maturity Model Integration/Capability Maturity Model.
5. Goal-Question-Metric.

Challenges facing software development process:

- Process design optimization or redesign
- Continuous process improvement (learning)

4.3.2 Process Discovery

Related to [process mining](#) is a set of techniques that automatically construct a representation of an organization's current business processes and its major process variations in order to extract what type of processes events, conditions, timestamps, and other meta-data from software development artifacts.

4.4 OSS Development Models

There are several basic differences between OSSD and traditional methods. Firstly, OSS systems are built by large numbers of people, largely volunteers. Secondly, work is not assigned; rather individuals choose to participate in specific project activities. Thirdly, there is no clear design process, at either a system or detailed level [31]. In addition, there is no explicit project plan, list of deliverables or schedule; all these differences suggest a weakening of traditional process models to be applied in OSS development.

Each OSS project has several processes and practices which have been followed and varied from each other based on the issues considered to developed it, this variation of the practices depend on the corporation developed the OSS products [32].

4.4.1 OSSD Project Characteristics

There are many features that characterize OSSD when it compared wit the traditional software development TSD [32]:

- OSS Developers are always users of what they build, while OSS users (>1%) are also OSS developers, where TSD differentiate between them only small or a limited team responsible of development activities.

- OSS requires “critical mass” of contributors and OSS components connected through socio-technical interaction networks to maintain a distributed development where not a compulsory issue in TSD is.
- OSSD teams use 10-50 OSSD tools to support their development work, which also not required in TSD.
-

4.4.2 Best Practices

There are some best practices that must be considered when trying to handle successful FOSS projects [33]:

- Processes with explicit process models are easier to manage, analyze, improve, distribute, and reuse.
- New/ reliable software tools and techniques are best candidates for software process support.
- OSSD is a community building process
 - not just a technical development process
 - FOSS peer review creates a *community of peers*
- OSSD processes often iterate daily versus infrequent singular (milestone) Software Life Cycle Engineering events
 - OSSD: frequent, rapid cycle time (easier to improve) vs.
 - SLC: infrequent, slow cycle time (harder to improve)
- Process management and improvement have been one of the most enduring practices in Software Engineering for improving productivity and quality, and to reducing cost and risks.

4.5 Existing OSS Development Models

Several researchers have proposed life cycle models derived from analyses of successful open source projects, the opinions differ as to the stages that comprise a

typical open source development project. However the OSSD paradigm demonstrates several common attributes [⁸]:

- Parallel development and peer review.
- Prompt feedback to user and developer contributions.
- Highly talented developers.
- Parallel debugging.
- User involvement rapid release times.

The coming part will describe existing OSS models that categorized to three types of models.

4.5.1 Comparative Model

This model have been suggested by Patrick Vixie in 1999 when he discussed that classic OSS projects such as BSD, BIND and SendMail are evidence of open source projects utilize standard software engineering processes of analysis, design, implementation and support [³²], which mean that an open source project can include all the elements of a traditional SDLC.

However, in his comparison between OSSD and the traditional SDLC, Vixie recognizes the fundamental differences that the OSS life cycle present such as code sharing and accessing, distributed code contribution and reviewing that often may appear in SDLC, but fails to suggest an appropriate model that analyses this new process.

4.5.2 Organizational Models

Schweik and Semenov in 2003 proposed an OSSD project life cycle comprising three phases: project initiation, going ‘open’, and project growth stability or decline. Each phase is characterized by a distinct set of activities.

Project initiation show that developers decide to take on projects for a variety of reasons; also it is premised on modularity, such that future development is organized around small manageable pieces. The advantages are: multiple programmers can work on the same module; competition for the best solution code increases quality; and there is greater control over project progress [33]

Going ‘open’ involves a choice on the part of the project founders to follow OSS licensing principles. In this phase appropriate technologies and web sites need to be chosen to act as a vehicle for sharing code and recruiting developers.

The final phase, **growth stability or decline**, poses an element of risk for open source projects: will the project generate enough interest to attract developers and users globally to use the product and participate in further programming, testing or documentation [7].

4.5.3 Task-related Models

Several researchers have derived life cycle models from investigating successful open source projects such as Apache and the FreeBSD Project; one of those researches is Mockus who described a life cycle that combines a decision-making framework with task-related project phases.

Niels Jorgensen has suggested a model that provides a more detailed description of specific product related activities that underpin the OSSD process. The below figure explains the life cycle for changes that occurred within the FreeBSD project.

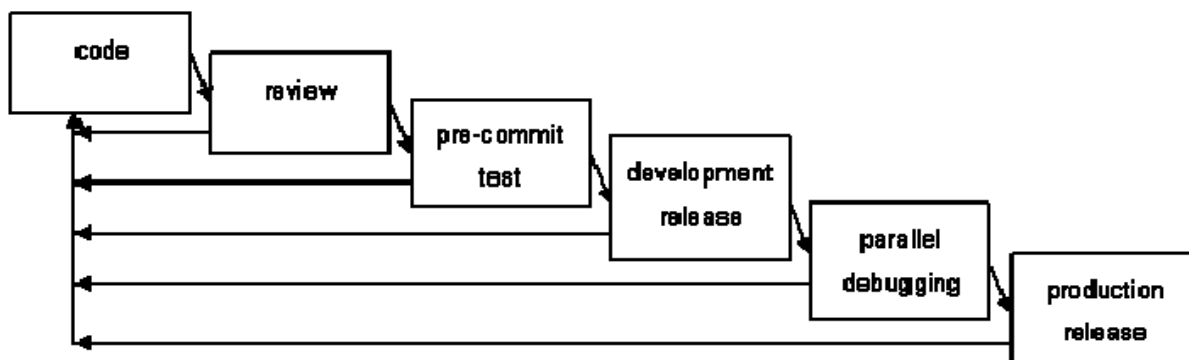


Figure 4.3: Jorgensen life-cycle [34]

There are several stages or sets of activities are proposed:

Code: Code is submitted by talented developers for review and improvement by respected peers.

Review: Most (if not all) code contributions are reviewed, this independent peer review is a central strength of the OSSD process.

Pre-commit test: Review is followed by an unplanned, yet thorough, testing of all contributions for a particular code change.

Development release: If the code segment is deemed release-ready it may be added into the development release.

Parallel debugging: Development releases of software perform a rigorous debugging phase where any number of developers is able to scrutinize the code in search of flaws.

Production release: Where development versions are deemed stable, they are released as production versions.

Jorgensen's model is widely accepted [35,11] as a framework for the OSSD process, on both macro (project) and micro (component or code segment) levels.

However, when applied to an OSS project, the model does not adequately explain where or how the processes of planning, analysis and design take place.

4.6 Proposed Framework

The proposed FOSS project development framework tends to identify the main steps that could be followed to adopt or handle such kind of projects working as a management framework; it takes into account the best practices that must be followed when trying to handle successful FOSS projects. It aims to identify what to do with little bit explanation telling how to do, working as a boundary that could be used to develop FOSS project within. Figure 4.5 explain the interactions involve within FOSS framework, generally the framework consists of two parts; OSS activities that may be

applied and FOSS framework actors. However, each OSS project can follow special development framework based on number of factors such as the vendor goals, the functionality, acceptance of the project itself (popularity reflect on the number of the developers who could participate), sponsors, technical issues, etc.

The suggested framework have been inspired from Sun Microsystems OSS development strategies of (NetBeens-OpenOffice) as partial work to meet the needs of developing FOSS software in order to fit the needs of adopting such project by firstly funded organizations or groups of individual developers who has sponsor.

4.6.1 Framework Objectives

The framework has several objectives aims to satisfy, including:

1. Identify what to do rather than telling how to do.
2. Encourage the adoption of FOSS projects.
3. Increase the productivity of the developers.
4. Facilitating testing process and bugs fixing.
5. Provide cheapest software and high quality software.

4.6.2 FOSS Framework Considerations

The suggested FOSS development framework corresponding to the most OSS development frameworks which are basically based on two considerations:

1. Distributed development work that performed on a completely volunteers developers environments as a social interactions between community members with some restriction control panel to ensure the success.
2. Preparing suitable technical infrastructure and providing communication tools such as web site, mailing lists, version control, bug tracking and real time chat to support the accessing, communicating, maintaining, Assigning and performing development work.

4.6.3 FOSS Framework Activities

1. As introduced before the framework consists from two parts; OSS activities that may be applied and the framework actors who could participate in such development, the framework provides a description of major product related activities as a template to start such development with ability to handle more miner specific activities assigned later during development which make it easily to adapt it., here are major activities that must be performed on such project after preparing the suitable technical infrastructure which is standard set of tools for managing information:
2. Write up a clear mission statement.
3. Choose a good name.
4. Identify the features and requirements list of the new projects.
5. Project announcements.
6. Provide a development status page, listing the project's near-term goals and needs.
7. Assigning and performing development work.
8. Preparing downloads.
9. Choosing a license and applying it.
10. Reviewing code, [testing and releasing](#).
11. Inspections.
12. Managing Releases.
13. Packaging.

Practically, there are more steps that could be added, or each step of those above includes a set of sub steps and activities.

As indicated preparing the suitable technical infrastructure is the basic assumption which all the development work is stand on, standard set of tools for managing information such as:

Web site: Primarily a centralized, one-way channel of information from the project out to the public. The web site may also serve as an administrative interface for other project tools.

Mailing lists: Usually the most active communications forum in the project.

Version control: Enables developers to manage code changes conveniently, including reverting and "change porting". Enables everyone to watch what's happening to the code.

Bug tracking: Enables developers to keep track of what they're working on, coordinate with each other, and plan releases. Enables everyone to query the status of bugs and record information about particular bugs. Also can be used for tracking not only bugs, but also tasks, releases, new features, etc.

Real-time chat: A place for quick, lightweight discussions and question/answer exchanges. Not always archived completely.

Each tool in those may have a distinct need, but their functions are also interrelated, and the tools must be made to work together, the next part will discuss the framework actors who are simply the people who contribute in project development and must cooperate with each other to satisfy their goal; developing a successfully FOSS project.

4.6.4 Framework Actors

The FOSS framework actors consist of nine collaborative elements (participants) as seen in figure 4.5 each participant work as standalone element on specific domain to avoid the roles interference:

- The Board.
- Community Manager.
- Users.
- Developers and Contributors.

- Quality Insurance Team.
- Release Manager.
- Maintainer.
- Site Administrator.
- CVS Manager.

To adopt such framework by large organizations or corporations it may be better to start the development work by those suggested roles as paid staff members responsible from managing other volunteer contributors and act as leaders of each part of the project. In case of adopting by small group of individuals it may be better to combine some roles together to reduce the total number of needed staff members because these projects will be developed by a self motivated developers with no large fund or sponsor who are simply small team, thereby it may be better to combine the board and the Community manger in one role; combing the Quality Insurance Team and the Maintainers in one role; combing the Release Manager and the CVS Managers in one role. However it will cause an extra overhead of each resulted role's responsibilities and expected conflict, to avoid such situation the basic assumption of the framework is to start working with all those roles.

4.6.5 Roles and Responsibilities

Each participant work as stand alone domain and has some activities to perform and share common tools with the other participants, here we will describe how actually the work must be done and what kind of roles to be played of each participant as follow:

4.6.5.1 The Board

The Board is always are free open source project's idea representative mainly their focus is to ensure that the project community is being run in a fair and open manner and make decisions for the community, on high level.

4.6.5.2 Community Manager

He is the person who shares the knowledge with the other participants and ensures that all community issues are addressed (respond to tech issues, unanswered questions) the community manager must be the most expertise person in the development team. Community manager cooperate with other contributors to maintain a roadmap document that specifies what will be included in future releases, as well as dates for which releases are scheduled. Also he determines content and timing, but goes to considerable to ensure that the development community is able to comment on and participate in these decisions.

4.6.5.3 Users

The FOSS users play the important role on such projects development, viewing users as co developers leads to code improvement and effective debugging because their comment and feedback reflect "what is going on the project" and "what can be done to resolve the project problems," so if encouraged, users can assist developers in finding system faults and improvements, thereby reducing the need (and cost) for extra developers to perform the same function. They always download and use free software, later on could be developers.

4.6.5.4 Site Administrator

Each FOSS project must have a web site which is the place for representing the organization or the group related issues, a web site manager is responsible from managing the website content (news, updates, etc) to ensure the site remains up-to-date and deploy builds.

4.6.5.5 Maintainer

The maintainer is the person who responsible from maintaining a project/ module, manages a group of developers, deciding features for the project, merge patches/bug fixes and create module web page.

4.6.5.6 CVS Manager

Each project provides an output at every time of adding new improvement as product releases, CVS manager or Concurrent Versions System manager is a person that who is responsible of managing, configuring, grant access and maintains CVS.

4.6.5.7 Developers

A developer is a person that contributes to community by selecting feature to develop, bug to fix, download, and commit code. A talent developer always meets the time constraints for the release and responds effectively to new ideas.

4.6.5.8 Release Manager

Release Manager starts new release phase by release proposal, release updates, branch for current release, release post mortem, review release candidates and decide final release, also must propose schedule/plan for the project.

The releases of any FOSS product are best illustrated by a tree of release branches with many branches where each major branch represents a major version; minor versions are represented by branches of the major branches.

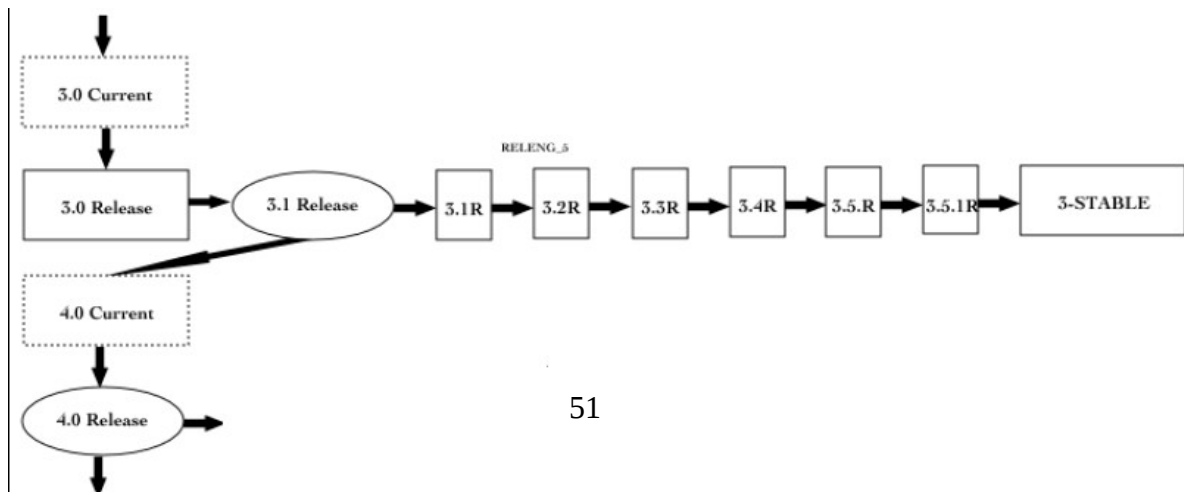


Figure 4.4: Releases Tree

In the following release tree, arrows that follow one-another in a particular direction represent a branch. Boxes with full lines and diamonds represent official releases. Boxes with dotted lines represent the development branch at that time. Security branches are represented by ovals.

4.6.5.9 Quality Assurance Team

Produce quality - builds and ensure quality of the software by downloading development builds and test to release Q-builds. Having a public forum for reporting problems and actually getting users and developer feedback on them is one of the most important advantages that identify the whole quality assurance process.

The main activities quality assurance are testing and bug fixing, there are many testing activities could be performed: ad-hoc volunteer testing, partial tests (not as complete as full functional tests) and full tests.

The suggested testing plan has two of testing types:

- Prerelease Testing

Performing a daily build, and runs a daily minimal “partial test” on the build, in order to ensure the build is sufficiently stable to allow development work on it to proceed.

- Inspections

Maintaining full test before releasing any version by both QA team (full) and Ad hoc volunteer contributors (partial).



Funds, support, Promote
Free/Open source Project

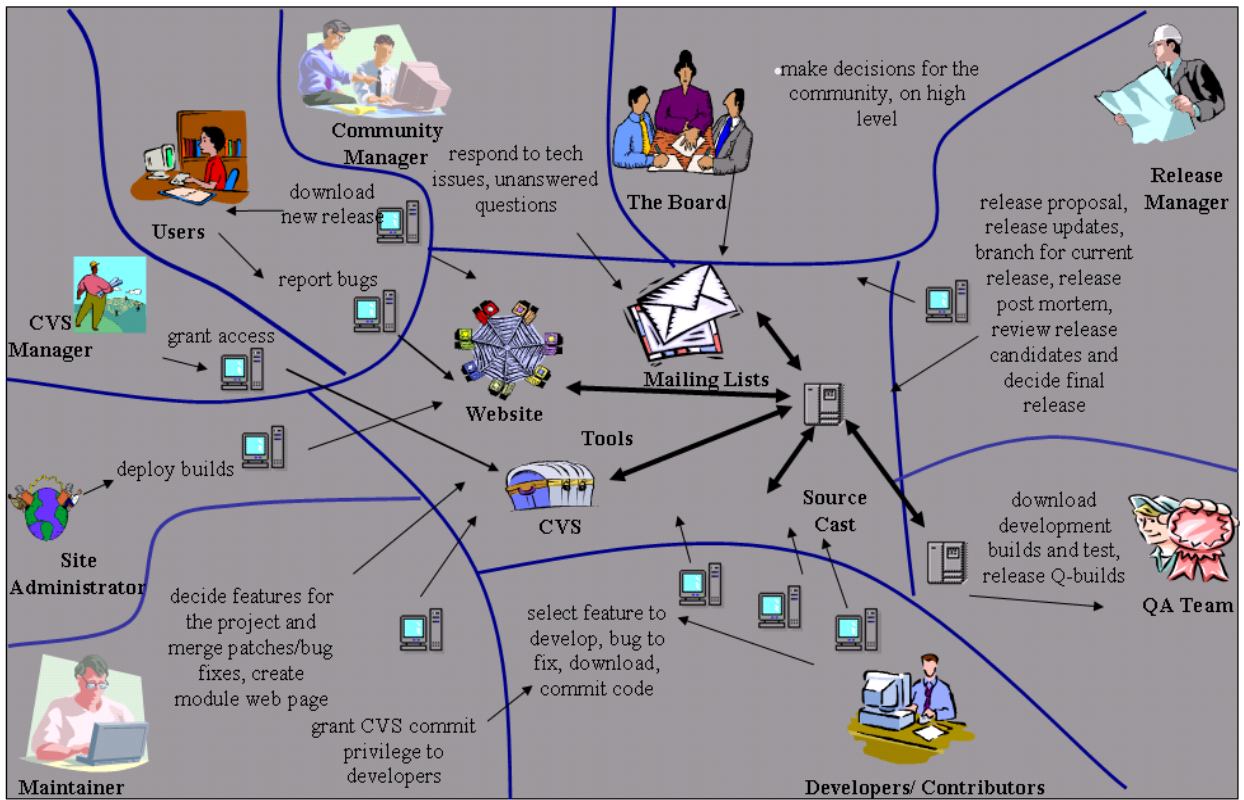


Figure 4.5: FOSS Framework

Here is the proposed framework diagram designed as Use Case diagram, in software engineering, a Use Case diagram in the Unified Modelling Language (UML), is a type of static structure diagram that describes sequence of actions.

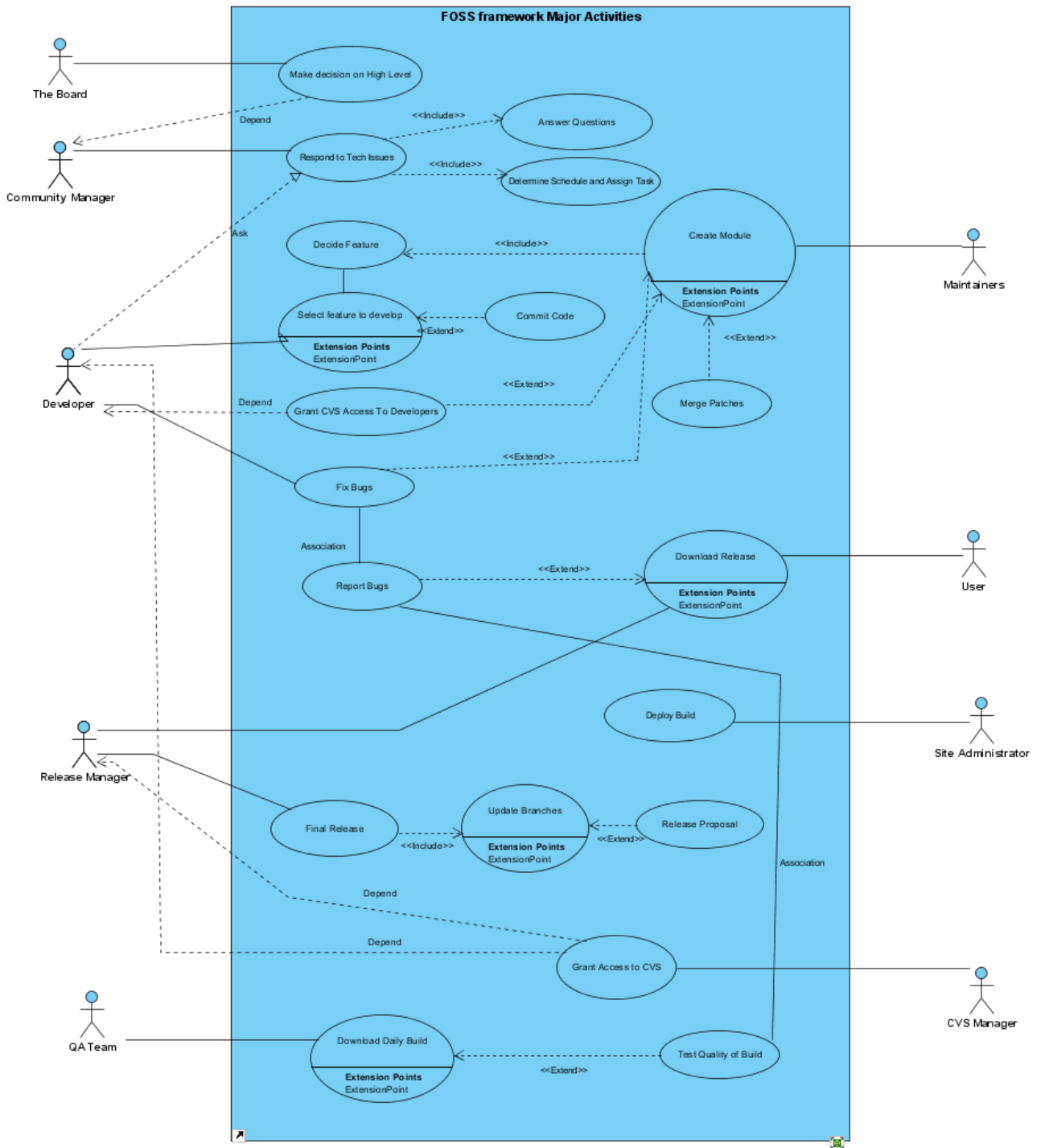


Figure 4.6: FOSS Framework

Chapter Five

Case Study of Mozilla Web Browser

5.1 Introduction

The Mozilla project is an Open Source Software (OSS) project which is dedicated to developing the Mozilla web browser suite. Since its creation in 1998, the project has attracted thousands of participants, and has arguably one of the largest communities working on an OSS project today [36].

Although its main product is the browser, the Mozilla Project has a number of related subprojects. The browser is developed using a set of open technologies which compose the Mozilla application framework, a platform-independent suite of languages and libraries, these technologies include:

- XUL, The XML User Interface Language, a cross-platform user interface description language
- XBL, the eXtensible Binding Language, a language used to modify the behavior of elements in documents
- JavaScript, an ECMA-standardized language for scripting Web applications
- Gecko, Mozilla's cross-platform, embeddable layout engine

5.1.1 Mozilla Organization and Community

The Mozilla Organization (mozilla.org) is a group which exists to support the development of the browser suite. Mozilla.org is responsible for managing, planning and providing server resources to support the development of Mozilla.

The organization is composed of selected people from the community who act as managers and technical lead for the various Mozilla projects. Each member of the organization is responsible for a Mozilla-related task, including Web site maintenance, documentation, architecture design and release engineering. There are currently 14 people listed as mozilla.org staff from a number of different organizations, including Netscape and Redhat [37].

Mozilla.org is charged with leadership for the Mozilla project, but it is important to realize that the actual work is performed by a large number of people who are not necessarily part of the organization itself, which are identified simply as the Mozilla community. The community consists of volunteers, paid contributors and mozilla.org staff.

5.1.2 Unique Aspects of Mozilla Project

Although the number of active OSS projects today is quite large [38], the Mozilla Project is an interesting target for OSS research for a number of reasons:

- The Mozilla codebase is one of the largest and fastest moving among OSS projects, its size is comparable to the Linux kernel.
- The original Netscape Navigator 5 codebase was donated by Netscape to mozilla.org [39], so there was a significant amount of pre-existing code at the time the Open Source project was officially started.
- The number of developers is high[35], many of them being directly paid by Netscape, OEone, Sun, IBM and other companies that fund development of the browser suite and framework.

	July	Aug	Sep	Oct	Nov	Dec	Jan	Feb(incomplete)
Developers submitting code	143	160	152	157	158	150	159	104
New developers submitting code	2	11	7	5	16	5	12	1
Total code submissions	1577	1892	1997	2355	2348	1594	2083	466

Table 5.1: Statistics for source code submission from July 2002 to February 2003[35].

- The Mozilla Project aims to create a polished, easy-to-use application for end-users of widely varying computer skills, whereas the majority of OSS projects concentrate on applications where the developer is also a domain expert.

5.2 Aspects of Mozilla Software Process

The work started on Mozilla in March 1998, using the original codebases which was donated by Netscape Communications. Because of this many of the requirements had been determined by the original code and documentation developed by Netscape [40].

However, the technical lead of mozilla.org came to the somewhat controversial conclusion that the original codebase would prove impossible to evolve to suit the requirements of a standards-compliant Web browser. Some code modules were completely rewritten – such as the layout engine, which needed to be thoroughly changed to support the new technologies.

5.2.1 Modularity and Module Ownership

The Mozilla browser is developed using the Mozilla application framework, one of the characteristics of the design of this framework is that it is highly componentized [41] due to the use of a cross-platform component library called XPCOM [42]. This design is by nature modular, and parts such as the Javascript engine, the runtime libraries, and the framework itself can be reused independently of the browser to develop other products. The high modularity also permits developers to concentrate on areas of the code without needing to understand the entire architecture; also it allows for a gradual learning curve, which is important for project newcomers.

The product is broken down into directories /layout, /mailnews, and so on. Files required to build a browser and mail reader are distributed among them (so that each one has an owner)

Most code modules in Mozilla have one or more associated components in the Bugzilla bug tracking tool. Each Bugzilla component has an owner, which is the default assignee for new issues reported, and a Quality Assurance contact.

5.2.2 “Bug-driven” Development

In the Mozilla Project the term **bug** is used to refer to any filed request for modification (MR) in the software, as a feedback of an actual defect, an enhancement, or a change in functionality. All change requests and their associated implementations have a unique number which identifies them in the Bugzilla problem tracking system.

Anyone can report bugs or request enhancements, the bug reporting and enhancement request process uses the Bugzilla problem-reporting tool, and requires requesters to set up an account on the system. Bugzilla also has tools that allow the bug reporter to see the most recent bugs, and if desired, to search the entire database of problem reports. Potential bug reporters are directed to use these tools to avoid duplicate bug reports.

Each bug is created with a state of UNCONFIRMED or NEW (depending on the experience the reporter is credited with). The task of actually confirming bugs by reproducing them rests on the volunteers who perform bug triage(fixing), which is one of the quality assurance activities in Mozilla.

5.2.3 Requirements

Often a controversial aspect of OSS projects [⁴³], the requirements process in Mozilla is also somewhat not clear because it was started by Netscape 5 features. High-level requirements are laid down by mozilla.org management, but since these are few and very abstract.

Most of the decisions on functionality inclusion and change are discussed piecemeal by the community and module owners through bug and newsgroup comments, a message thread is started on a public newsgroup, regarding a change in functionality. Other people will usually comment on relevance and discuss advantages and disadvantages.

It is hard to say that the requirements process is generally inadequate: the community has active participation in the adoption of proposed features, and anyone is free to implement a desired change and submit it for approval. So the module owners and mozilla.org staff are the final authorities for determining and approving these changes (level of control that the Mozilla process requires).

5.2.4 Design

The actual process of designing the Mozilla software architecture is difficult to abstract because of two important issues: first, the design inherits in part from original Netscape experience, so it was not completely invented in public view; second, because design discussions are inherently difficult to capture and usually have sparse record^[44].

According to Mike Shaver and Dan Mosedale, an engineer for Netscape, the original Mozilla design was a direct evolution of the Netscape Navigator 5 architecture.

5.2.5 Distributed Development and Formal Reviews

One of the premises the Mozilla Project was based upon was that face-to-face communication should not be required for development, which is strictly the rule for most, if not all, OSS projects^[45]. Thus all code would have to be designed, implemented, tested and integrated without relying on personal contact to solve problems, this poses many difficult situations and requires planning and support tools. All developers work using revision control (CVS) on a common, centralized, codebase, which allows changes to be developed concurrently and independently. There is a single image of the code, and at any time any developer can easily retrieve the “tip”, which is the latest version of the Mozilla source

Mozilla uses tools such as Bonsai and Tinderbox tools to provide a way to query in real time the status of the repository, and the most recent changes.

The review process works as follows: a developer working on a change for a bug produces a patch, which is a generated text file which describes the line-by-line differences made between the developer’s local version and the latest version in the code repository. This patch is then attached to a bug in the Bugzilla system, and the developer requests review. A reviewer, which can be the module owner or anyone else familiar with the code, will then read the code critically and either grant review or ask for changes.

5.3 Issues Related To Mozilla Framework

Mozilla framework is a decision-making framework with task-related project phases but failing to explore how analysis and design phases start from the beginning because these phases inherits in part from original Netscape experience, so it was not completely invented in public view.

As we discussed that Mozilla is currently operated by the mozilla.org staff (14 members) that coordinate and guide the project, provide process, and engage in some coding. Only about 4 of the core members spend a significant part of their time writing code for the browser application. Others have roles dedicated to such things as community QA, milestone releases, Web site tools and maintenance, and tools such as Bugzilla that assist developers.

Mozilla .org concentrates on the following areas as phases to develop their browser:

- Roles and Responsibilities.
- Identifying work to be done.
- Assigning and Performing Development Work.
- Prerelease Testing.
- Inspections.
- Managing Releases.

5.4 Mozilla vs FOSS Framework

In order to evaluate the suggested FOSS framework a comparison done between the two frameworks that show the major similarities and differences between them in term of conceptual factors, furthermore the entire OSS frameworks seems to be similar on most of the development work practices with differences of how to manage the work and organize the team.

5.4.1 Actors and Roles

The Mozilla.org staff member currently consist from 14 people who distributed between deferent assigned roles as we discussed that in section 5.1.1, 5.3.1 there is a board who act as managers and technical lead for the various Mozilla projects,

responsible from ensuring that the project community is being run in a fair and open manner and make decisions for the community, on high level which is typically as we described in the suggested framework.

Also there is web site administrator who is responsible from managing the website content, release manager who is responsible from release engineering activities, QA Team who are responsible from ensuring the product quality.

However the Mozilla.org staff member is not specifically conform the suggested hierarchy because it is base on the actual work responsibilities, some tasks performed by hired staff from other corporation, here we see that our hierarchy identified nine key roles depending on self motivated staff members work as a team and distribute the roles responsibilities between them.

Another point discussed that the actual work is performed by a large number of people who are not necessarily part of the organization itself, which could be identified simply as the project community. This community consists of volunteers and paid contributors as we did in the suggested framework with some attentions to excite great numbers of participants because we considered completely full FOSS criteria on developing, distributing and integrations.

5.4.2 Quality Assurance

Quality assurance activities performed by different classes of people, ranging from QA engineers and volunteers for both frameworks with some differences on the way to perform each test level strategies.

The main activities QA in both frameworks considered several levels of quality as test plans:

- Prerelease Testing.
- Inspections

Mozilla have a term to refer for partial test called smoke test (not as complete as full functional tests)

5.4.3 Similarities and Differences

Similarities	Mozilla Framework	FOSS Framework
Face-to-face Communication	Not required	Not required
Distributed Environment	Exists	Exists
# of Key Roles	9	9
Communication Tools	Yes	Yes
Quality Assurance Phases	Prerelease Testing. Inspections	Prerelease Testing. Inspections
Modularity and Modules Ownership	Yes	Yes
Category	a decision-making framework with task-related project phases	a decision-making framework with task-related project phases
Major OSS Development steps	Yes	Yes
	Mission statement Identify the features and requirements Project announcements. Reviewing code Testing Inspections Releasing Packaging	
Differences		
Operability	Large corporation and organizations fund in OSS	Small organizations and groups of individuals with their sponsor

Quality Assurance Levels	Two Levels (Smoke Test – Full Test)	Three Levels (Partial-Ad hoc- Full Test)
Releases	Small pieces of upgrades	Full Software updates or complete new version
Requirements	Partial View	Public view
Design	Partial View	Public view
Project Management	Mozilla Staff Leads the entire development	Virtual Project Management (VPM)
Abstraction level	high detailed level of it's steps	Low detailed level of it's steps
Development Projects	OSS	FOSS
License	Mozilla Project License (MPL)	General Public License (GPL)

5.4.4 Findings

So it will be clear that the suggested FOSS framework has the following features which could make it better in developing such kind of projects.

- The framework was designed to meet the needs of small groups and organization that adopting FOSS projects with ability to be adopted in the commercial environments to produce products with fee charge, it's hierarchical structures allow adding new sponsor or investors at the top of the hierarchy.
- The framework takes into account developing completely FOSS as primary objective, which will help to satisfy the other framework objectives because when considering completely full FOSS criteria on developing, distributing and integrations you will gain increasable growing fast community which increase the productivity of the developers and facilitate the testing process and bugs fixing.
- Each participant work as stand alone element on specific domain to avoid the roles interference.

- The Framework Identify key roles with clear role responsibilities which make it more powerful.
- Mozilla framework didn't separate between the maintainers and testers related tasks each one of them could perform a testing plan which could affect the quality assurance consistency, in contrast with FOSS framework we find that quality assurance activities done by specified team who is responsible from the entire testing plan, no conflicts with maintainer's responsibilities which is bounded with just bugs fixing activities and other related tasks.
- FOSS framework provide three Quality Assurance levels partial, full test levels that performed by maintainers and ad hoc volunteer test which could make it more accurate when it compared with Mozilla framework that provide only two levels of test.
- Mozilla framework is a decision-making framework with task-related project phases but failing to explore how analysis and design phases start from the beginning because these phases inherits in part from original Netscape experience, so it was not completely invented in public view, where the FOSS framework is also a decision-making framework with task-related project phases with a clear defined analysis and design phases because it's invented for public view.
- FOSS Framework tend to be more flexible when it provides a description of major product related activities as a template to start such development with ability to handle minor specific activities assigned later during development which make it easily to adapt it.

Chapter Six

Conclusion and Recommendations for Future Works

6.1 Conclusion

Free and Open Source Software or FOSS offers the world of software industry much better things dispersing from concepts, methods, techniques, etc. when it compared with the proprietary software applications which are strictly protected through patents and intellectual property rights

This research has issues of concerns; provide of the main issues and facts about free and open source software projects nature and open source software development processes. The basic principle for the OSS development process (OSSDP) is that by sharing source code, developers cooperate under a model of systematic peer-review, and take advantage of parallel debugging that leads to innovation and rapid advancement.

The proposed FOSS framework which consists of two parts; OSS activities that may be applied and FOSS framework actors summarized the best practices of developing OSS projects based on some successfully OSS projects which could be followed to develop FOSS projects by small organizations and groups of individuals. Then we present a case study of Mozilla Web browser as a successful OSS project with attention to catch the similarities and differences between Mozilla framework and the suggested FOSS framework in order to provide a validation of proposed framework.

6.2 Recommendations for Future Works

Practically, FOSS framework provides a low detailed level of the entire FOSS project development steps, there are more steps that could be added, or each step of those mentioned could include a set of sub steps and activities it is recommended to provide details of those sub steps and their associated activities.

In order to evaluate the proposed FOSS framework a comparison done between the two frameworks that show major similarities and differences between them in term of conceptual factors, furthermore Mozilla framework seems to be similar to FOSS framework in different areas, it recommended to make a validation with other different existing frameworks such as apache web server .

Future researches could validate and evaluate the FOSS framework with ability to enhance or adding new part.

Another suggestion is that studies must be done to compare between FOSS Projects and property software project in term of adoption overhead is it better to adopt FOSS project in the developing countries or continue buying and using property software.

6.3 References

¹[] E.E. Kim, “An Introduction to Open Source Communities, Blue Oxen Associates, Technical report,” 2003.

²[] Hansen, H.,R., and Janko, W, Wirtschaftsf, “Results from Software Engineering Research into Open Source Development Projects, ” university, Austria, 2000.

³[] A. W. Brown, and G. Booch, “Reusing Open-Source Software and Practices: The Impact of Open-Source on Commercial Vendors,” Springer- Verlag, 2002.

⁴[] Cary Sullivan, “Managing Open Source Projects,” 2001.

⁵[] Eric S. Raymond, “[The Cathedral & the Bazaar](#),” O'Reilly. ISBN 1-56592-724-9, 1999.
<http://www.catb.org/~esr/writings/cathedral-bazaar/cathedral-bazaar/>, [viewed on May 2009].

⁶

[] Walt Scacchi, “When Is Free/Open Source Software Development Faster, Better, and Cheaper than Software Engineering,” Institute for Software Research University of California, 2006.

⁷[] Open Source Initiative website, <http://www.OSI.org> , [viewed on July 2009].

⁸[] Rinette Roets, Mary Lou Minnaar, Kerry Wright, “Open Source: Towards Successful Systems Development Projects in Developing Countries,” University of Fort Hare, 2006.

⁹

[] Walt Scacchi, Free and Open Source Development Practices in the Game Community,”University of California, Irvine, 2005.

¹⁰[] Asiri, S, “Open source software,” Computers and Society, 2003.

¹¹[] FLOSS Project Report, Free/Libre and open source software: Survey and study, 2004.

¹²[] Walt Scacchi, “Opportunities and Challenges for Modeling and Simulating Free/Open Source Software Processes,” Institute for Software Research University of California, 2004.

¹³[] W. Scacchi, “Understanding the Requirements for Developing Open Source Software Systems,” 2000.

¹⁴[] <http://www.starcraft2.com> , [viewed on Oct 2009].

¹⁵[] <http://www.bnetd.org> , [viewed on July 2009].

¹⁶[] K. Fogel, “Open Source Development with CVS,” Coriolis Press, 1999.

¹⁷[] R.T. Fielding, “Shared Leadership in the Apache Project,” Comm. ACM, vol. 42, 1999.

¹⁸[] <http://www.quakeforge.net> , [viewed on Aug 2009].

¹⁹[] <http://www.mame.net> , [viewed on Aug 2009].

²⁰[] A.J. Kim, Community-Building on the Web: Secret Strategies for Successful Online Communities, Peachpit Press, 2000.

²¹

[] <http://www.PlaneShift.it> , [viewed on Aug 2009].

22

[] Madey, G., Freeh, V., and Tynan, R., “Modeling the F/OSS Community: A Quantative Investigation, in S. Koch (ed.),”Hershey, PA, 2004.

23

[] Hertel, G., Neidner, S., and Hermann, S., “Motivation of software developers in Open Source projects: an Internet-based survey of contributors to the Linux kernel, Research Policy,” July 2003.

24

[] Stewart, K.J. and Gosain, S. , “ An Exploratory Study of Ideology and Trust in Open Source Development Groups, Proc. 22nd Intern. Conf. Information Systems,New Orleans, LA. 2001.

25

[] Ye, Y. & Kishida, K., “Towards an understanding of the motivation of open source software developers,”Portland, OR, IEEE Computer Society, May 2003.

26

[] Monge, P.R., Fulk, J., Kalman, M.E., Flanagan, A.J., Parnassa, C., and Rumsey, S., “Production of Collective Action in Alliance-Based Interorganizational Communication and Information Systems,” Organization Science, 1998.

27

[] Espinosa, J. A., Kraut, R.E., Slaughter, S. A., Lerch, J. F., Herbsleb, J. D., Mockus, “ A shared Mental Models, Familiarity, and Coordination: A Multi-method Study of Distributed Software Teams,” Barcelona, Spain, December 2002.

²⁸[] http://www.softpanorama.org/SE/software_life_cycle_models.shtml , [viewed on September 2009].

29

[] McConnell, “Rapid Development, Microsoft Press,”1996.

30

[] A Spiral Model of Software Development and Enhancement May 1988.

³¹[] P.Vixie, “Open sources: Voices from the open source revolution,” California, 1999.

³²[] Walt Scacchi, “Process and Open Source Software,” Institute for Software Research, October 2008.

³³[] Schweik, C. M., & Semenov, The institutional design of open source programming: Implications for addressing complex public policy and management problems, 2003.

³⁴[] Jorgensen, “Putting it all in the trunk: Incremental software development in the FreeBSD open source project,” Information Systems Journal, 2001.

35

[] Feller, J., & Fitzgerald, “Understanding open source software development,” London: Addison-Wesley, 2001.

³⁶[] The Mozilla Organization. mozilla.org at a glance,<http://www.mozilla.org/mozorg.html>, [viewed on Oct 2009].

37

[] The Mozilla Organization. mozilla.org Staff Members, <http://www.mozilla.org/about/stafflist.html>, [viewed on Oct 2009].

³⁸[] Freshmeat.net. Statistics and Top 20, <http://freshmeat.net/stats> , [viewed on Aug 2009].

³⁹[] Frank Hecker. Mozilla at One. 1999, [http://mozilla.org/mozilla at-one.html](http://mozilla.org/mozilla-at-one.html) , [viewed on Oct 2009].

⁴⁰[] The Mozilla Organization. Documentation Graveyard, <http://www.mozilla.org/classic/>, [viewed on Oct 2009].

⁴¹[] Surveys of user and developer participation in the project to gather general satisfaction and perceived problems with relation to the Mozilla, <http://www.mozilla.org>, [viewed on Oct 2009].

⁴²[] The Mozilla Organization. XPCOM. 2001, <http://www.mozilla.org/projects/xpcom> , [viewed on Oct 2009].

⁴³

[] Lisa GR Henderson., “Requirements Elicitation in Open-Source Programs,”.2000.

⁴⁴[] Thomas R. Gruber and Daniel M. Russell, “Design Knowledge and Design Rationale: A Framework for Representation,” Knowledge Systems Laboratory, Stanford University, 1990.

⁴⁵

[] Audris Mockus., “Roy Fielding, and James Herbsleb., June 2000.

[⁴⁵] Linton, J., “Process Mapping and Design: ebruari 2007.

[⁴⁶] www.Wikipedia.org, [viewed on Nov 2009].