Quality Assurance in Open Source Software Development by Introducing Requirement Gathering Phase

Thesis submitted in partial fulfillment of the requirements for the award of degree of

Master of Technology
in
Computer Science and Applications

Submitted By
Navneet Chanalia
(Roll No. 601003016)

Under the supervision of
Ms. Vineeta Bassi
Assistant Professor, SMCA

SCHOOL OF MATHEMATICS AND COMPUTER APPLICATIONS
THAPAR UNIVERSITY
PATIALA – 147004
July 2012

CERTIFICATE

I hereby certify that the work which is being presented in the thesis entitled, "*Quality Assurance in Open Source Software Development by Introducing Requirement Gathering Phase*", in partial fulfillment of the requirements for the award of degree of Master of Technology in *Computer Science and Applications* submitted in School of Mathematics and Computer Applications of Thapar University, Patiala, is an authentic record of my own work carried out under the supervision of *Ms. Vineeta Bassi* and refers other researcher's work which are duly listed in the reference section.

The matter presented in the thesis has not been submitted for award of any other degree of this or any other University.

Signature: Navneet.

(Navneet Chanalia)

This is to certify that the above statement made by the candidate is correct and true to the best of my knowledge.

Vineeta Bassi 03/07/2012

(Ms. Vineeta Bassi)
Assistant Professor,
SMCA

Countersigned by

(Dr. S.S.Bhatia)
Head
School of Mathematics and Computer Applications
Thapar University
Patiala

(Dr. S. K. Mohapatra)
Dean (Academic Affairs)
Thapar University
Patiala

ABSTRACT

Most of the software engineering concepts that one studies are written in the context of Closed Source Softwares (CSS) development and not in the context of Open Source Software (OSS) development. So we strongly believe that there should be a study to improve the quality of OSS because they save a lot of money of the end users. A report by the Standish Group states that adoption of open-source software models has resulted in savings of about $60 billion per year to consumers. So, with this thought this thesis studies the various aspects and facts of open source software developments. The main objective of this thesis is to find out the gaps in the current open source software development process. This thesis also implements a way to cover those gaps. And hence provides a framework to develop better open source softwares in future.

ACKNOWLEDGEMENT

First of all, I would like to express my gratitude to **Ms. Vineeta Bassi, Assistant Professor,** School of Mathematics and Computer Applications (SMCA), Thapar University, Patiala for her patient guidance and support throughout this thesis work. I am truly very fortunate to have the opportunity to work with her.

I am also thankful to **Dr. S.S.Bhatia, Head of SMCA,** as well as **Sh. Singara Sir, Assistant Professor,** SMCA, entire faculty and staff of SMCA. I am also thankful to friends who have devoted their valuable time and helped me in all possible ways towards successful completion of this work. I thank all those who have contributed directly or indirectly to this work.

Navneet.

Navneet Chanalia
(601003016)

List of contents Page No.

List of Figures Page No.

List of Tables Page No.

List of Abbreviations

CSS	Closed Source Softwares
SDLC	Software Development Life Cycle
OSS	Open Source Softwares
PC	Personal Computer
BSD	Berkeley Software Distribution
GPL	GNU Public License
LGPL	Lesser General Public License
MIT	Massachusetts Institute of Technology
MPL	Mozilla Public License
PQM	Project Quality Management
SQA	Software Quality Assurance
CMM	Capability Maturity Model
EMR	Electronic Medical Record

Chapter 1

INTRODUCTION

Closed Source Softwares (CSS) are the softwares those are developed by developers working under the rules and regulations of their employers. These employers are various national and multinational software development companies. All these companies have their own predefined software development methodology. All the software developers working under such companies have to follow a certain set of Software Development Life Cycle (SDLC).

But the same is not true in case of Open Source Software (OSS) Development. OSS are developed by volunteer developers for free. There are no predefined SDLC in case of OSSD. Also there is neither quality assurance nor requirement gathering phase in OSSD. Due to these reasons end users are highly skeptical about adopting and considering OSS as a viable alternative to closed source software.

So this thesis work includes the study of various aspects of OSS development and finds out the reasons behind lack of quality and users skepticism about the open source software. This thesis will introduce an open source software development model and a requirement gathering portal to cover up the gap between the OSS development and end users. So as to, assure the end users that the developed OSS will satisfy their needs and requirements.

This chapter gives an introduction to open source softwares and quality assurance.

1.1 Open Source Software (OSS)

Open Source Software (OSS) is software that is developed by volunteer developers for free. It also includes its source code along with the executable file of the software. In this way OSS are free to use and modifiable.

There are two major ways in which work on an open source project can be initiated:

1. An individual who senses the need for a project announces the intent to develop the project in public. The individual may receive offers of help from others. The group may then proceed to work on the code.

2. A developer working on a limited but working code releases it to the public as the first version of an open-source program. The developer continues to work on improving it, and possibly is joined by other developers.

So we can say open source software development is a kind of distributed software development that has a large amount of contributors and because of using Internet and sharing idea freely it is so successful and useful for developers to communicate over the distance. Usually the idea of starting an open source project comes from a draft idea or existing closed source software with personal motivations, needs or philosophy beliefs or to get portion of the market. Moreover for some individual participant it can be useful to measure their skills and capabilities. However it also can be a freedom of choices against huge previous business product such as Microsoft or IBM's which are controlling the software world or even comes from free thinkers or people who are not interested in making money but contributing to the well-being of the society [1].

1.2 History of Open Source Software

More than 50 years ago, when the first mainframes came to the market and were sold by firms hardware and software integrated together [2]. According to Hertel [3], in the 1960s, when the researchers start to use computers for their work, due to the lack of commercial software availability, they had to refer to software code open sharing at that time. In 1960s, the possibility to send emails supported this free exchange code sources. Around 1969 IBM began to unbundle (separating hardware and software) and software became an independent product and was attended more and more. In 1981, the modification in the patent law in the United State and the emergence of the PC (Personal Computer) caused to the source code closing of the UNIX operating system [2]. In

September 1983, Richard Stallman launched the GNU project (that initiated GNU operating system) to make a free UNIX-like operating system and he initiated the free software movement. He also pioneered copy left licenses like GNU General Public Licenses that is the most widely used free software license [4].

Open Source, then opened its place within the hackers–the people who love programming or do it for fun, or even for creating viruses [5].

List of some popular and successful open source softwares:

1. Android.
2. Apache web server.
3. Open Office.
4. Mozilla Firefox.
5. MySql.
6. LINUX.

1.3 Current Scenario

Today companies are looking for a methodology to support and maintain their software. This is because of increasing the attention of supporting and protecting software, after the development, rather than only building the code. Therefore open source methodology due to having special characteristics plays a vital role to satisfy customers and companies as well. Unlike traditional methods which define teams and requirements, the processes in OSSD model is parallel and repeatedly development techniques, with free user participants, huge development communities and effective user testing.

Moreover due to involvement of many people to develop, test and evaluate the code, OSSD gets more benefits rather than traditional closed source software. Although OSSD model has some problems to challenge, but still is a proper methodology especially for large scale software companies.

1.4 Open Source versus Closed Source Software

Table 1.1

Comparison of Open and Closed Source Softwares [6].

PROPERTIES OF CLOSED SOURCE VERSUS OPEN SOURCE

Type / Property	Closed Source	Open Source
Copyright	Yes	No
Sold	Yes	No
Available Source Code	No	Yes
Free Software	No	Yes
Modify Code by Users	No	Yes
Systematic Development	Yes	No
Frequent Release	No	Yes
Openness	No	Yes
Parallel Process	No	Yes
Knowledge Sharing	No	Yes
Private License	Yes	No
Gaining Profit by Company	Yes	No

- Closed source softwares are copyrighted and are developed to gain profit after selling them. On the other hand open source softwares are developed by volunteer developers for free.
- The best feature of the open source softwares is that they come with their source code. So that we can view and modify the source code according to our requirements. In this way user can enhance the functionality of open source software. But same is not true in case of closed source softwares.

- Another property of open source software is that their new versions are released very frequently. But in case of closed source softwares the end users have to wait longer to receive working software.
- In traditional development model, the software is protected by copyrights and is sold to users to earn money, while in open source paradigm; the software is available to users freely to use and to change it.
- In traditional way, development of software is a systematic and formal work that is performed by skilled and expert developers to enhance the quality. While in OSSD model software is improved by the feedbacks of users and developers around the world. There are no special people or place to sit and discuss physically. In OSSD model, the improvement of quality of codes may depends on the interest and skills of each individual who participates through registering in some host sites like *sourceforg.net* and it retains to the responsibility of each person against system to work better and enhance the quality.
- The successful issue in quality of closed source is related to centralized management while in open source the quality is from its openness because many programmers examine it and can detect bugs. It also will be reevaluated more and more. Although it can be caused to have iterative correction and may some developers correct a same thing. Moreover the quality in OSSD model is improved rather than closed source because the process of developing software in open source such as reviewing, testing and maintainability performed parallel.
- The target of closed source is software development to present to the public for gaining the market penetration and profit earning. While open source is a personal idea that can be spread with joining the other hands from all over the worlds to write and develop it, though coordinating of this large scale efforts, needs to manage it.
- There is no ownership for open source and it belongs to the public while closed source belongs to one software firm and all the benefits will reverse to that company, but it is also possible to abuse from free codes or available ideas.

1.5 Problems in Using Open Source Softwares

In case of collaboration problems, since communication between developers should be very close in software development, and in open source it is only through the internet, therefore the geographically distance and distributed collaboration leads to decrease communication among developers. Another problem is related to users that contribute the project until the end or not because there is no obligation to do and they are completely free. The release management policy demonstration is troublesome, because from the start of project, the release management guidelines are informal. Moreover when we do not have adequate contributors or interested users in OSS projects, it may have a negative effect on the motivation of volunteers, to keep maintaining the project. In addition any patch submitted to the project must pass the peer review process, in which code style, general quality and the interoperability with other parts of the application, are assessed [7]. Waring [8] notes the drawbacks of the open source software for business such as version proliferation that is the same problem with closed source, complex and numerous licenses, implementation issues that there is no single person is responsible for developing and resolving the problem of software, high short-term costs for OSS unlike the long-term costs that is lower than traditional model. Furthermore the economical challenge of open source to motivate volunteers to participate without money is also considerable [8]. Another disadvantage of using open source will clear, when we lose internet or a fighting issue starts among programmers, which can hurt developing [9]. Moreover it may happen in open source, which for a long time no updates for the system; and have to work with older version, since there is no certain responsibility to give the regular update for the software [10]. There is a similar issue for open source to support as well, because there is no support at all for open source unlike closed source, through a single company, especially, if we faced to an emergency problem that needs a quick answer [11]. In large OSS projects, large contributions can take longer to review, which can be problematic for volunteer developers [12].

1.6 Open Source Software Licenses

One may wonder what is the need of license for open source software when it is free and open.

It is true that software can be made OSS by releasing it uncopyrighted and there after the user of this OSS can view, modify and release it again. But problem will occur if any of the new users of this program releases it as a copyrighted version because in that scenario the new version of the actual software is not open source any more.

To avoid this scenario open source licenses use the concept of 'copyleft' which states that anyone who redistributes the software must pass on the freedom to further copy and change it. So this is one of the reasons why licenses are used with open source softwares.

If software is released under an OSS license then the software must be used and distributed as specified by the license.

Some of the OSS licenses [13] are given below:

1. **Berkeley Software Distribution (BSD)** - It was introduced in 1970 and was originally written at the University of California Berkeley. It is used to allow commercial use of the software.
2. **GNU Public License (GPL)** - It was introduced in 1991 and it was written by Richard Stallman Source code of OSS under this license can be used, viewed and modified but cannot be marketed. It also forbids programs to link to the exclusive rights.
3. **Lesser General Public License (LGPL)** - It was published in 1991 by free software foundation. Unlike GPL it allows links to a library or non LGPL/GPL program.
4. **Massachusetts Institute of Technology (MIT)** - It was written at the Massachusetts Institute of Technology. MIT permits OSS to be used under exclusive rights.

5. **Mozilla Public License (MPL)** - MPL was introduced at the Netscape Communications Corporation and is used for Mozilla Firefox, Mozilla Application Suite, and Mozilla Thunderbird. MPL is a combination of the BSD and GPL licenses.

1.7 Software Quality

According to Roger Pressmen [14] *software quality* is defined as conformance to explicitly stated functional and performance requirements, explicitly documented development standards, and implicit characteristics that are expected of all professionally developed software.

1. Software requirements are the foundation from which quality is measured. Lack of conformance to requirements is lack of quality.

2. Specified standards define a set of development criteria that guide the manner in which software is engineered. If the criteria are not followed, lack of quality will almost surely result.

1.8 Software Quality Engineering

Software quality engineering includes three very basic, but important steps [15]:

- Define a software engineering process.
- Assure adherence to the process.
- Improve the process.

1.8.1 Software Process

A software process defines the common "world map" and terminology for those working in a software project to achieve the same goal. The process can be viewed as "borders" within which the project execution "path" has to go. A specific/detailed process gives a

narrow “road” and a less detailed process gives a wider one. The definition of processes for a project is a typical task for the quality management group in the project.

1.8.2 Process Adherence

If the project does not adhere to the process, the project “path” goes outside the process “borders”. This is a major problem from a quality point of view. Firstly, it is hard to analyze and improve the process if the project using it neither adhere to the process definition nor document the deviations from the process. Secondly, the activities depicted in the process are all aimed at developing good quality software in an efficient manner. If the process is not followed, for example, if some mandatory documents are left out, the quality of the product will be affected. To identify this type of problems, activities for process adherence are defined.

The goals for Software Quality Assurance (SQA) group are threefold:

- Monitor the software and the development process.
- Ensure compliance with standards and procedures.
- Report needs for improvement to manager`s attention.

It is however important to note that neither the SQA activities nor other quality activities are aimed at “adding the quality” to the software product. The quality is being built into the product as defined by the processes. The task of SQA is measuring and monitoring the processes used by the project. As stated by the Capability Maturity Model (CMM) “The purpose of Software Quality Assurance is to provide management with appropriate visibility into the process being used by the software project and the projects being built”. Another key issue is that problems identified are reported and acted upon as close to the source as possible. The SQA personnel should act as independent observers which report problems to the actor/developer and the actor/developer has to decide upon and take actions to solve the problem. This is well in line with the principle of letting each role take the responsibility for quality.

1.8.3 Process Improvement

As a result of SQA activities the development process may have to be improved. Process improvement means in terms of our metaphor, that the borders are moved. However, if the development does not adhere to the process, the borders can be moved but the execution is not affected at all. Hence the process improvement initiative does not help. The first step in process improvement is to ensure that the process and its application are conform, i.e. the project execution path falls within the borders, or deviations from the path are under control. In this case, the process can be improved by moving the borders. The need for process improvement can be identified by the SQA, but the effort spent in doing the improvement is a task for the project and its quality management.

1.9 SQA Organization

The SQA group is an independent observer which shall identify and bring to attention where conformance is lacking between the stated process and the actual execution. In order to be independent, the SQA group has to belong to an organizational unit, separated from the software project to be audited. However the SQA group must work close to the project in order to monitor it. A suitable organizational chart is shown in figure 2.1.

The SQA group shall report to senior management in order to have enough power to escalate problems if appropriate measures are not taken. On the other hand, the organizational level of the SQA personnel shall not be too high, since this creates distance between SQA group and the staff.

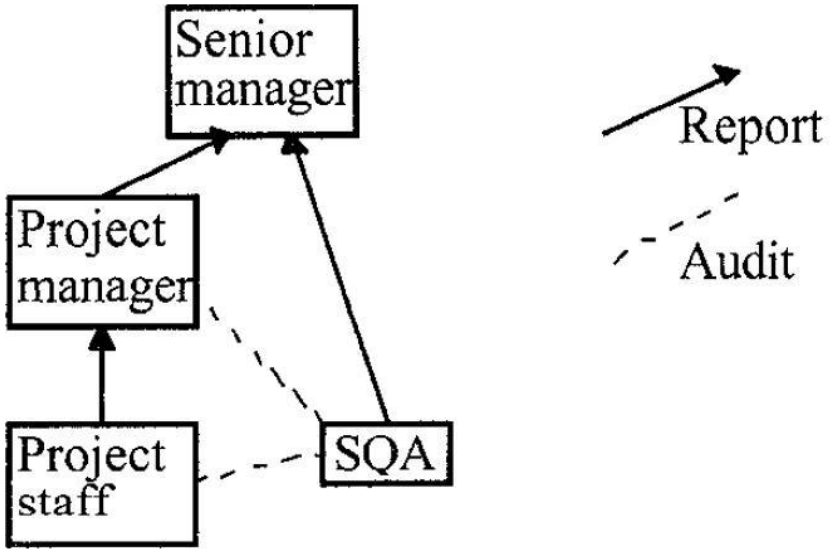

Figure 2.1 SQA Organization

1.10 SQA Activities

There are three techniques for the SQA [14]:

Reviews - inspection of project documents and taking part in project reviews.

Audits - planned and spontaneous audits to check adherence to working procedures and to verify project progress.

Measurements - draw conclusions on process adherence based on measurements. Note that the SQA group is not collecting data, but using raw and analyzed data.

Activities to be performed by an independent SQA group are:

- Review development and quality plans for completeness.
- Participate as moderator in design and code inspections.
- Review test plans.
- Review a sample of test results to determine adherence to plans.

- Periodically audit software configuration management to determine adherence to plans.
- Participate in project phase reviews.

The list above will in practice create numerous of reviews and audits to be performed by the SQA group. Hence a sampling has to take place.

Examples of sampling methods are:

- Ensure that inspections take place and take part in a selected set.
- Review inspection reports.
- Ensure that tests are performed and test reports are written and examine a set, preferably those which fall outside of established control limits.

1.11 Tools and Techniques of Quality Assurance

1. Benefit / cost analysis

The planning process must consider benefit/cost tradeoffs. It is elementary that the benefit should outweigh the cost

- **The Primary Benefit:** is less work, higher productivity, lower costs, and increased stakeholder satisfaction.

- **The Primary Cost:** is the expenses associated with Project Quality Management (PQM) activities.

2. Benchmarking

Benchmarking involves comparing actual or planned project practices to those of other projects to generate ideas to:

- Generate ideas for improvement.
- Provide a standard for measurement of performance.

Other projects compared may be within the same organization or outside and may be within the same application area or in another.

3. Flow charting

- The flowcharting techniques in quality management generally includes:
    - cause and effect diagram
    - System or process flow charts
- Flowcharting can help in anticipating probable quality problems and thus helps to develop approaches for dealing with them.

4. Design of Experiments

- This is an analytical technique which aims to define variables that have most influence on the overall outcome
- This technique is commonly applicable to the product of the project issues.
- However this technique can also be used in project management issues such as cost and schedule tradeoffs to allow for optima solutions.

5. Quality Audits

Quality Audits are a structured review of other quality management activities: they may be carried out timely or randomly.

They may be carried out by properly trained Internal-auditors or by third parties such as quality systems registration agencies.

Chapter 2

LITERATURE SURVEY

2.1 Quality in terms of Open Source Software Development

Quality of OSS development relies on the following key areas:

- Sustainable communities.
- Code modularity.
- Peer Review.

To achieve high-quality software, OSS practitioners must fully understand these areas and how they relate to each other.

2.1.1 The Sustainable Community

High-quality OSS relies heavily on having a large, sustainable community to develop code rapidly, debug code effectively, and build new features. Many studies concluded that creating a sustainable community should be an OSS project's key objective. An investigation of OSS project evolution found that a large base of voluntary contributing members was one of the most important success factors. In particular, the study found that the system and the community must co-evolve for the software system to have sustainable development and achieve high quality [16].

The onion model:

In this model (see figure 2.1), the sustainable community consists of a small number of core developers and increasing numbers of contributing developers, bug reporters, and users.[16] "Onion" refers to the successive layers of member types. This is the most common model of a sustainable community. Individuals create the sustainable community by increasing their involvement through a process of role meritocracy. As they move toward the core, users become bug reporters and might over time become contributing developers. A few contributing developers will eventually join the small

team of core developers. Each type of member has certain responsibilities in the system's evolution, which relate to the system's overall quality. Advancement through the member types is reward and recognition for each member's abilities and achievements.

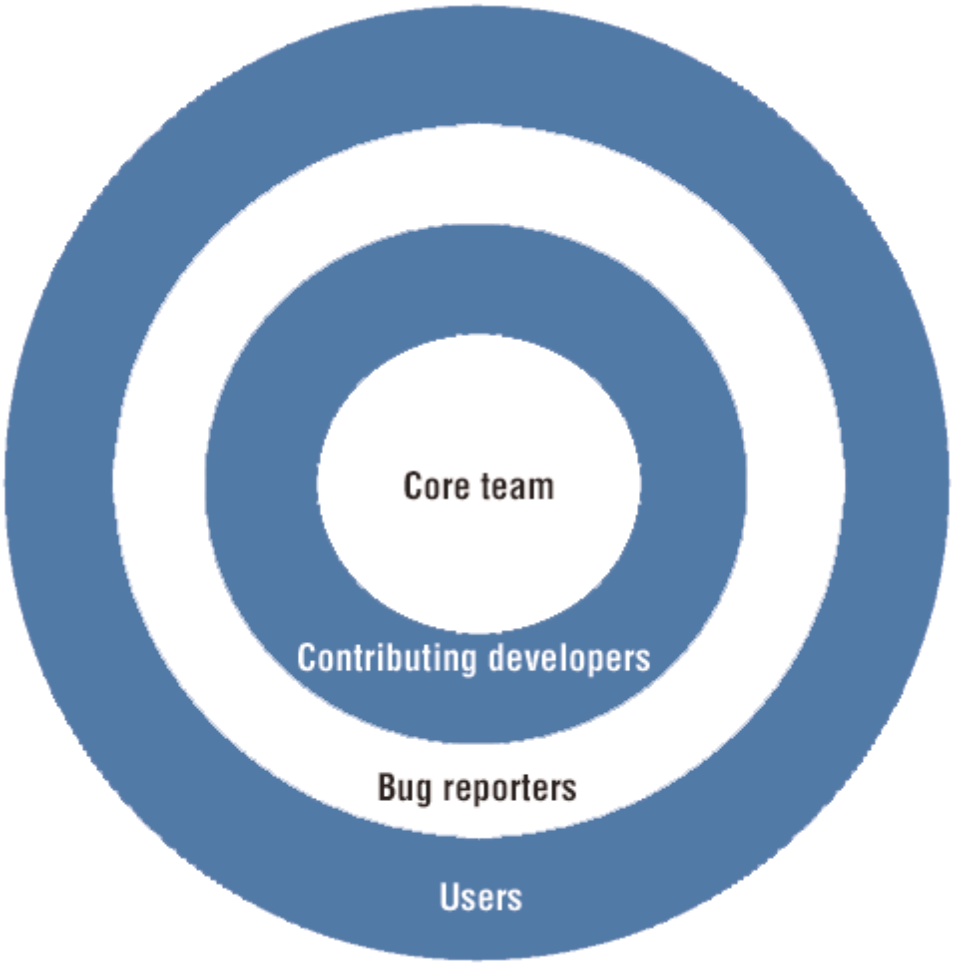

Figure 2.2 Onion Model of Sustainable Community for OSS development.

The onion model has three primary characteristics:

1. **The core team must be small**: As well as performing the bulk of the coding activity, the core team will exert control over the core system to maintain high modularity. They integrate only high-quality code, determine and follow the project roadmap, and maintain a fast, iterative release cycle. In a case study of Apache, a 15-person core development team performed over 80 percent of the functionality coding. Most core teams will be much smaller but will perform a similar amount of work.
2. **Contributing developers add and maintain features:** The ability to easily add new features depends on code modularity. Contributing developers tend to choose areas that are outlined in the project roadmap or that "scratch an itch." They'll also undertake bug fixes, peer-review code, and help ensure that the core team isn't overwhelmed with bug fixes.
3. **Bug reporters take ownership of system testing and bug reporting:**

   The core or contributing developers will do few o these tasks. The sheer size of the bug-reporting group will ensure that more people test the system (on more platforms) than any commercial organization could hope to achieve. This group plays a key role in reducing defect density.

Participation and motivation:

A sustainable community relies on attracting volunteer programmers. Much research has been done into what motivates the volunteer community. OSS managers must have a strong awareness of this area. While high code modularity reduces complexity and lets newcomers contribute peripherally without impacting the core system, modularity alone won't attract programmers to a project. In Audris Mockus and his colleague's case study of Mozilla—a highly modular project that initially had trouble attracting contributors participation increased only after the core team improved documentation, wrote tutorials, and refined development tools and processes. Other key motivators included

opportunities to learn new technologies and tools and to take on new challenges. OSS teams must understand that programmers rarely volunteer their services selflessly- they wish to gain status in the community through reward and recognition. Various studies have concluded that:

- Programmers are looking for social status, which is "determined not by what you control, but by what you give away," and so strive to develop high-quality code.
- Recognition and flattery by a community giving credit where it thinks it's due achieved by publicly naming contributors and letting people sign their own work helps keep contributors motivated.

As a project snowballs and becomes more successful, it will attract more developers who wish to bask in the project's glory and get some of the attention.

2.1.2 Code Modularity

Gwendolyn Lee and Robert Cole's study of the Linux kernel development further supported the premise that code modularity lets many programmers extend the program by working on separate modules, without needing to change or understand the core system, or interfere with each other's progress. This reduces the risk of new bugs being introduced in other modules. The study of the Apache and Mozilla projects also concluded that both projects' low defect densities resulted from employing high code modularity and many bug finders and fixers. The largest study to date on code modularity's impact on OSS quality investigated 100 open source C applications and established a clear relationship between high modularity and quality.

The study described how small component size derived from good design and resulted in low defect density and high user satisfaction and facilitated maintenance and evolution. It would be worthwhile for further research to focus on open source applications developed in other programming languages.

Another study of the Linux kernel development concluded that modularity let multiple developers work on the same solution, often in competition, increasing the probability of timely, high-quality solutions. Others have pointed out that bug fixing can also become a

competition to come up with the best, most efficient, and longest-term fix, so a high-quality fix tends to be the outcome.

2.1.3 Peer Review

OSS peer review assesses whether a contribution merits acceptance into the code base. Eric Raymond asserted that the rapid release cycle facilitates peer review because implementing and responding quickly to peer reviewers' comments and code keeps them involved and interested. This results in a product that grows and extends rapidly and reaches high quality quickly. Conversely**,** Steve McConnell noted that while large numbers of peer reviewers are clearly fast and effective, they aren't necessarily efficient and that best practice software engineering indicates five to six peer reviewers as optimal.

Raymond coined "given enough eyeballs, all bugs are shallow," meaning that if enough people see a software error, at least one of them will probably understand the error's causes and be able to fix it. This is particularly important given that a Linux kernel study concluded that 75 percent of the development work is typically mundane, labor-intensive tasks such as debugging, code reviews, and fixes.

2.2 Case Studies

(A) The study of 200 OSS projects discovered that:

- Fewer than 20 percent of OSS developers use test plans.
- Only 40 percent of projects use testing tools, although this increases when testing tool support is widely available for a language, such as Java.
- Less than 50 percent of OSS systems use code coverage concepts or tools.[17]

(B) According to one study, the user base performs the bulk of the system testing, sometimes even exclusively, as with Apache. A large pool of bug reporters is therefore crucial, ensuring software testing on many platforms. Others have concluded that OSS

generally has lower defect density than traditional software because the latter can't test as extensively. [18]

(C) A study of the Mozilla project by Christian Reis and Renata de Mattos Fortes showed that it had an unusually strict quality assurance process and dedicated test teams. [19]

Chapter 3

PROBLEM STATEMENT

3.1 Problem Definition

According to Roger Pressmen [14] – "software requirements are the foundation from which quality is measured. *Lack of conformance to requirements is lack of quality*".

As far as closed source software development is concerned there is a requirement gathering phase in every software development model. But in case of open source software there is *no defined phase of gathering the requirements from the end user* of the software. Results suggests that, rather than gathering needs from users, documenting those needs in a specification, then analyzing and validating the specification, open source *developers frequently assert the existence of requirements*, sometimes by supplying a complete implementation.

Hence we can say that there is a *negligible role of the end user* in the process of open source software development.

3.2 Justification

Whether software is open source software or closed source software it is finally used by the end user, not by the developer. So it becomes necessary to consult the end user before developing the software. Otherwise the end user will have to compromise on some features of the software.

A case study [20] of an open source software called as Electronic Medical Record (EMR) software has shown that 10 of the 13 features (i.e. 77%) were introduced by developers. Of these, 6 were introduced by a core developer. 4 of these were proposed by one individual, a consultant whose business is installing and tailoring OpenEMR for clients. The results have shown that, similar to Firefox, the majority of OpenEMR requirements

are introduced by developers; a few were proposed by users; none were influenced by competing products. Also validation of requirements and documentation is informal and typically involves an on-line discussion among developers.

Chapter 4

Proposed Solution and Implementation

As stated in the 'problem statement'- "lack of conformance to requirements is lack of quality."

So we can say that requirement elicitation is the foundation of good quality software. But, in open source development there is no phase to gather requirements from the end users.

Results [20] suggests that, rather than gathering needs from users, documenting those needs in a specification, then analyzing and validating the specification, open source developers frequently assert the existence of requirements, sometimes by supplying a complete implementation.

Hence, the absence of requirement gathering phase in OSS development is the basic reason behind the lack of quality in open source softwares. In other words there is a negligible role of the end users in the open source software development. So, if we want to assure the end users that open source softwares will fulfill their needs and satisfy their expectations then we have to introduce their role in development from the very first phase.

The gap between the OSS development and end users can be covered by collecting requirements from the end users prior to the development phase of OSS. However, we should always remember that the developers of open source softwares are volunteers. They are not paid for their work. Open source developers choose their work; they are not assigned any work.

Keeping the above points in mind we have proposed two solutions:

1. **Open Source Software Development Model-** This thesis has deduced a **framework** for the development process of open source softwares.

2. **Requirement gathering portal for OSS**- This thesis has proposed and developed an online portal to collect the actual requirements of the end users. The motive is to **introduce the requirement gathering phase** in the open source software development.

4.1 Open Source Softwares Development Model (Proposed)

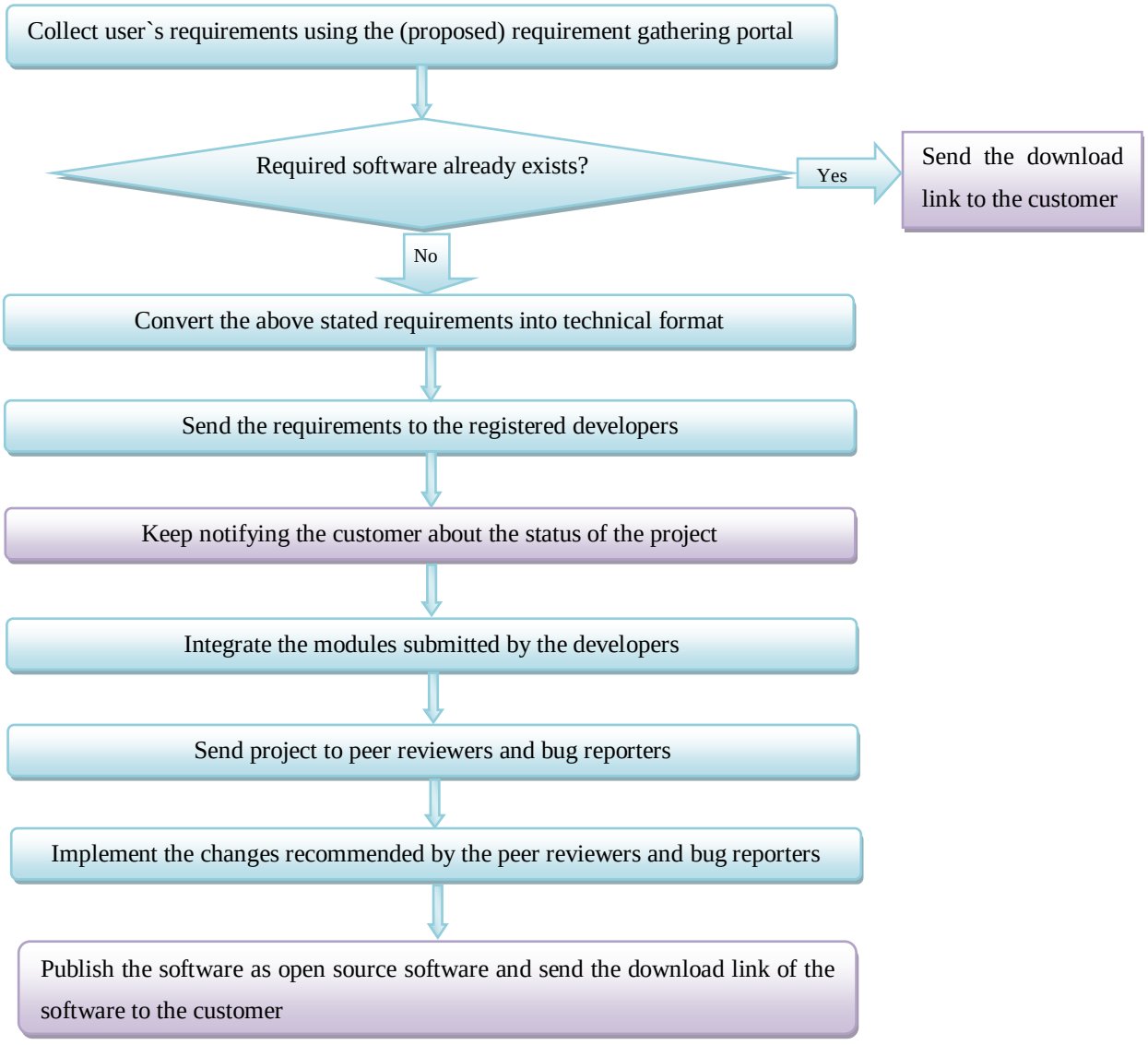

Figure 4.23 Proposed software development model for OSS.

The above shown model is the proposed development model for open source softwares. The purpose of developing this model is to introduce the requirement gathering phase in the OSS development and hence increasing the role of the end users and satisfying their actual software needs.

4.1.1 Terminology

1. **Requirement gathering portal**- It is a web portal proposed and developed by us to introduce the requirement gathering phase in the OSS development.
2. **Download link**- It is the internet download link/website of the required software.
3. **Registered developers**- It refers to the developers those are registered on our proposed requirement gathering portal.
4. **Customer**- Here customer is the end user who wants new and free software according to his requirements.
5. **Bug reporters/Peer Reviewers**- These are those members of the open source community who uses the software and report back about the quality, bugs and problems in the software.

4.1.2 Explanation of the Proposed Development Model

Step 1- Collect the new software requirement using the proposed requirement gathering portal. End users will register and submit their software requirements on the proposed requirement gathering portal.

Step 2- Check if the required software already exists. If there is already a software which is same as the required software then send the download link to the customer. Otherwise convert the non-technical requirements submitted by the customer into technical form.

Step 3- Send these requirements to the developers registered on the proposed requirement gathering portal.

Step 4- Integrate the modules submitted by the developers. If there is more than one module available for the same requirement then choose the best out of the available alternatives.

Step 5- Forward this integrated software to the bug reporters and the peer reviewers.

Step 6- Implement the changes recommended by the bug reporters and the peer reviewers.

Step 7- Publish the software as open source software and send the download link of the software to the customer.

4.2 Overview of Requirement Gathering Portal

If we take a look on the development process of closed source software then we can clearly observe that the role of the end user starts at the very first stage. But here in open source software development there is no role of the end user at all. So we need a system that could cover this gap between the development process of OSS and the end users.

So` we have proposed and developed a requirement gathering portal for the open source softwares. On this portal there will be three types of users:

- **Customer**
- **Moderator**
- **Developer**

Working of requirement gathering portal:

First of all interested customers and developers will have to register themself on this portal. There after the customers will post their new software requirements on this portal. These requirements will not be visible to the developers until they are modified into technical form by the moderator. Once the non technical requirements posted by the customer are modified into technical form they will be visible to the developers. All the modified requirements will be visible in the developer's inbox. Developers can also use the search button to search the software requirements of their interest. If the developer is interested in developing the new software then he can add that software into his undertaken project list. And once the software is completed developer will post its

download link on this portal. Moderator and customer can check the status of the developing software any time. Once the software is complete its download link will be available to the customer.

Four possible status of any software developing on this portal are:

- Is in queue
- Converted into technical form.
- Developers are working on it
- Developed

Tools used to develop the portal:

1. Microsoft Visual Web Developer Express Edition 2008
2. Microsoft SQL Server 2005
3. Internet Information Service (Windows XP)

4.3 Detailed Working of Requirement Gathering Portal

This section includes the following areas:

- Registration.
- Adding new software requirements.
- Modifying requirements into technical form.
- Development of the software.
- Status of software.
- Managing account.
- Database tables used.

Registration- First of all, if someone has new software requirements then he/she has to register himself/herself on our proposed requirement gathering portal as a customer.

Every customer has to choose a unique username. They will have to give their basic information like full name and email address.

In the same way interested OSS developers are also required to register on this portal. However, the moderator(s) of this requirement gathering portal will not be required to register on this portal. They will be provided with their login credentials directly.

Below is the screen shot of the registration form for the requirement gathering portal.

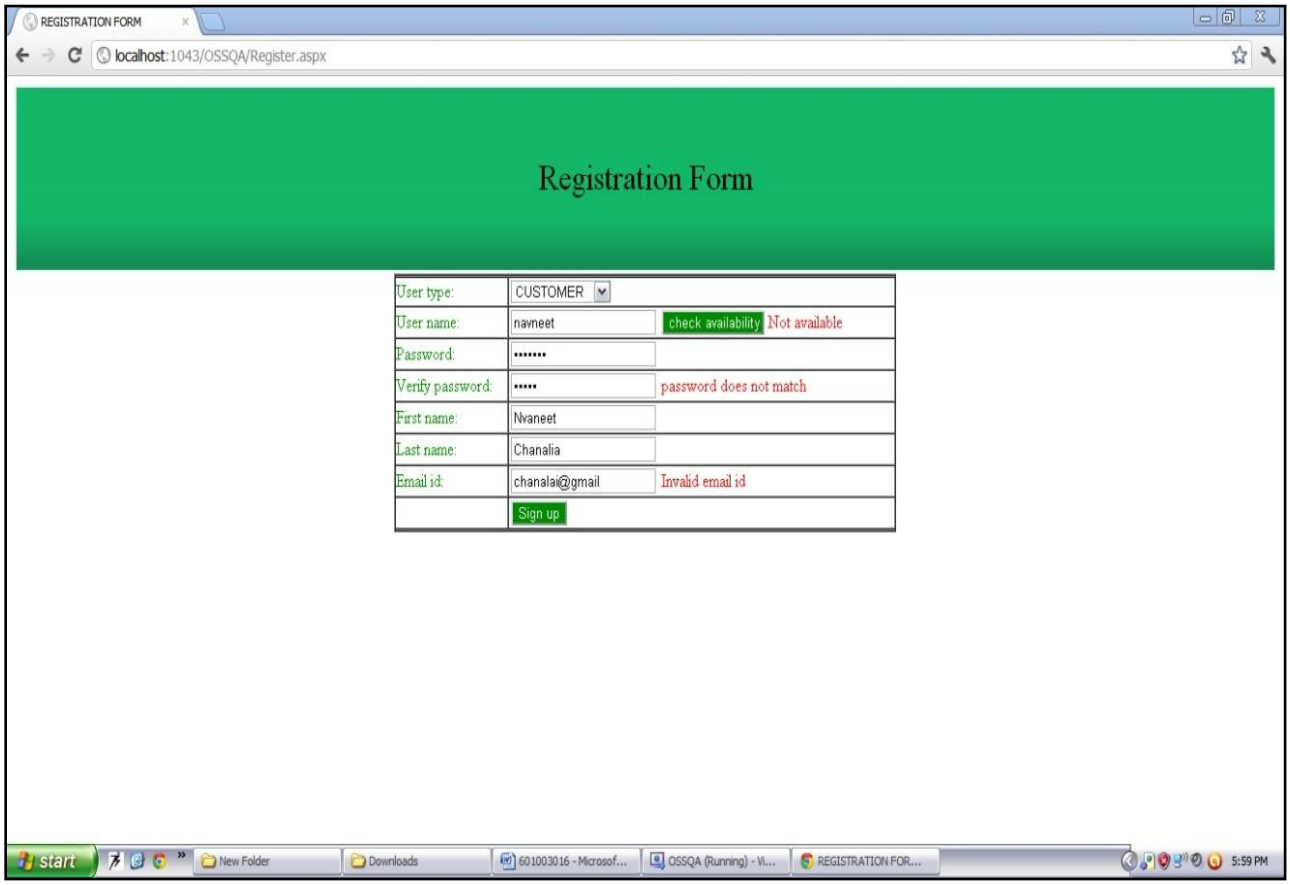

Figure 4.2 Registration Form.

Login Form-

Figure 4.3 Login Form.

As there are three types of users for this requirement gathering portal, so, every user will have to select the user type at the time of login. As shown below:

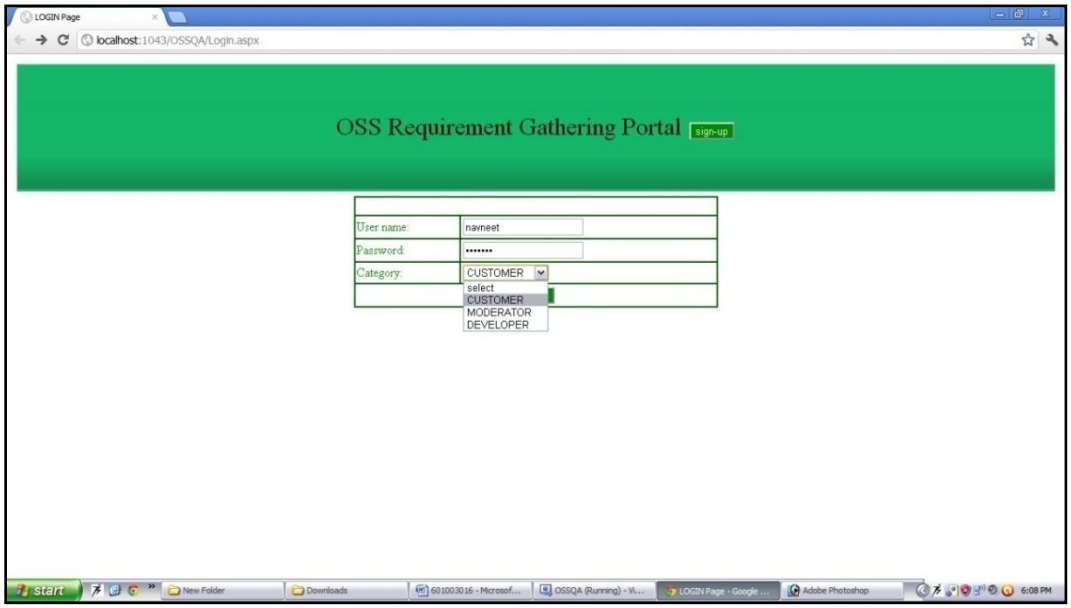

Figure 4.4 User`s category in login page.

Adding new software requirements-

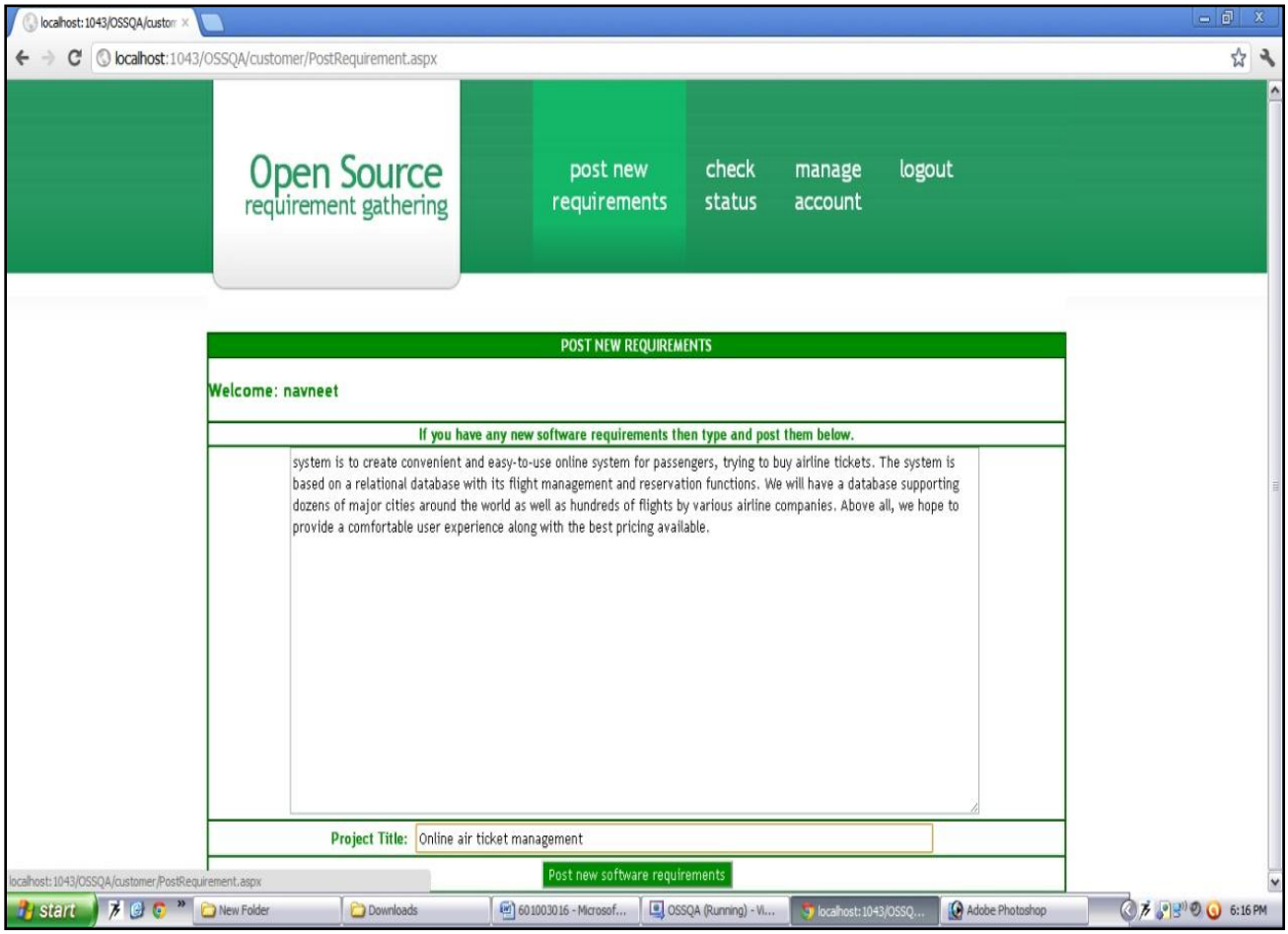

Figure 4.5 Post new software requirements.

Once a user is registered as a customer he/she can add introduce their software requirements by clicking the 'post new requirements' tab. The customer will also be required to provide a title for his/her requirements.

After posting the requirements a message will a message will be displayed to the customer. If any of the field is empty then the requirements will not be posted to the server and the required field valuator will come in action.

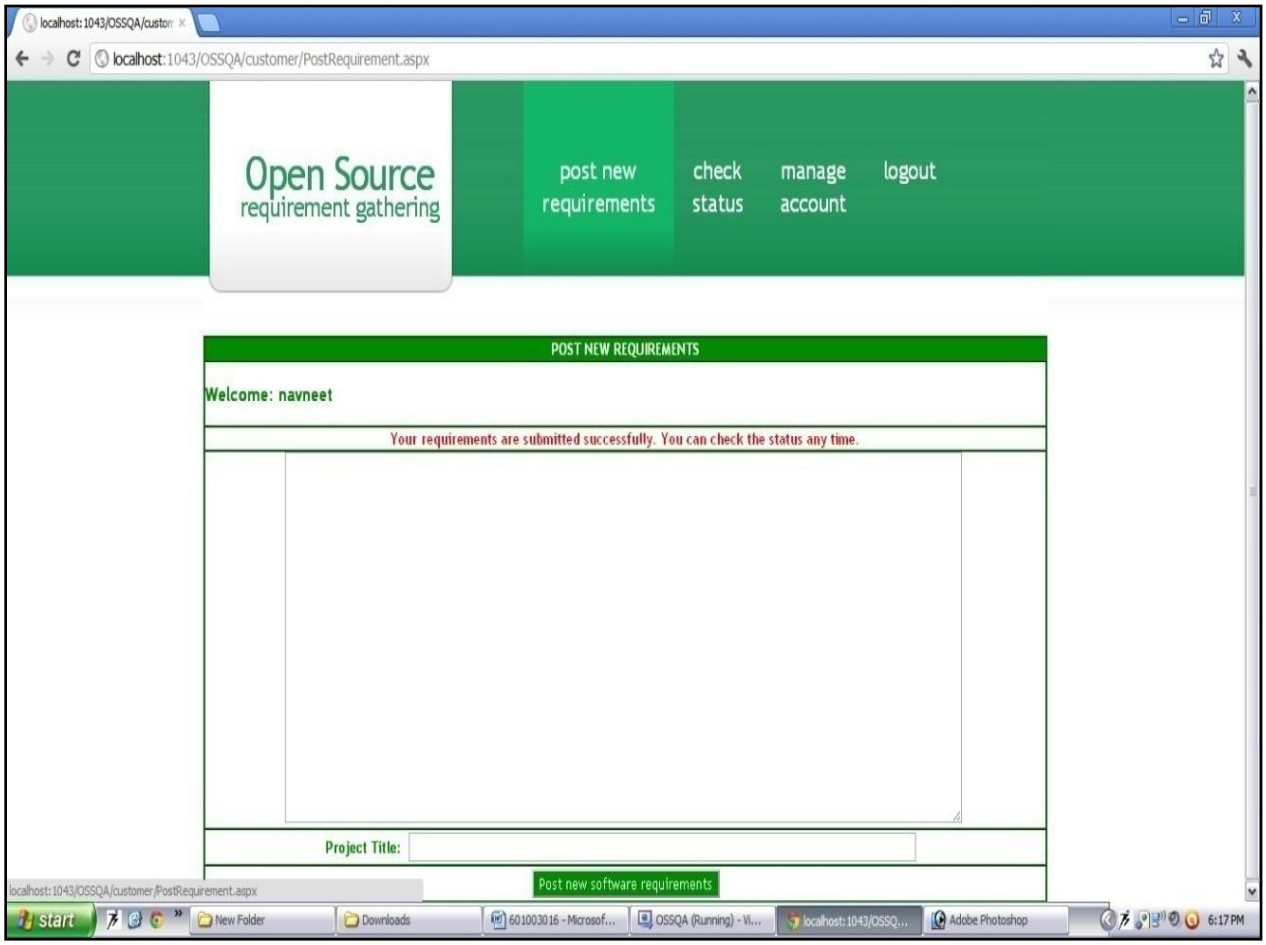

Figure 4.6 Requirements submitted successfully.

Modifying requirements into technical form-

It is not necessary that each and every requirement posted by the customers is in technical format, because every customer needs not to be a technical person. Keeping this point in mind we have introduced a user called as moderator for our proposed requirement gathering portal. The major role of this moderator is to convert the non-technical requirements into technical form. So that requirement could be easily understood by the developers. Every requirement posted by the customers will appear in the moderator`s inbox as shown below:

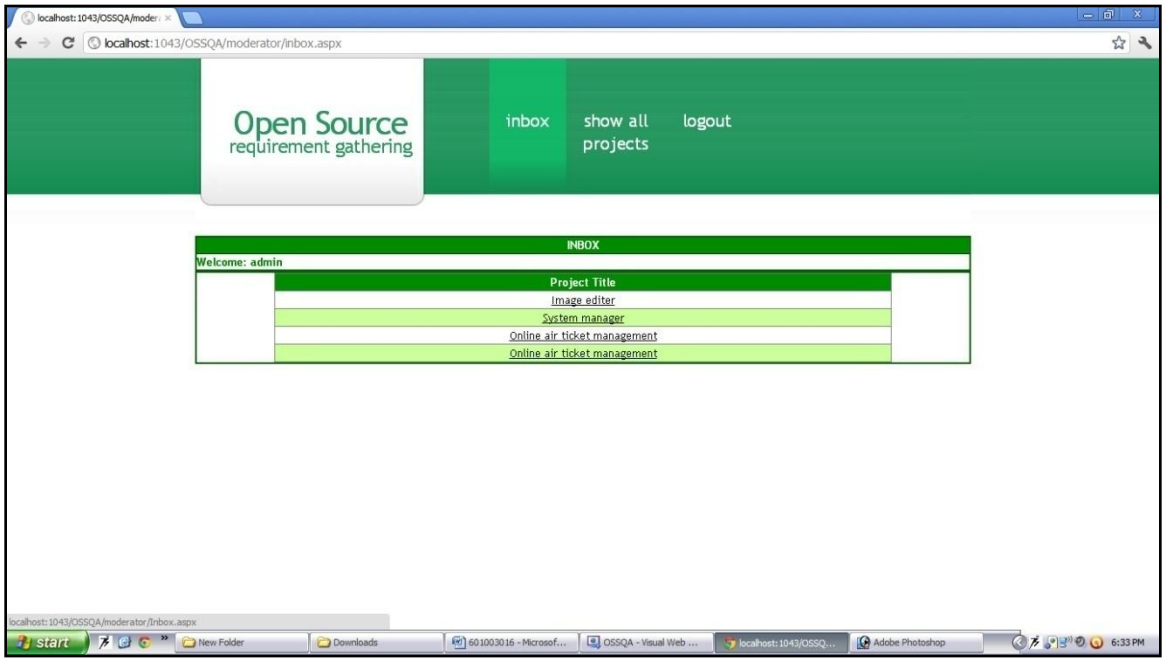

Figure 4.7 Moderator`s Inbox.

Each requirement can be viewed by clicking on it. When a requirement will be clicked in the moderator`s inbox it will be viewed as shown below:

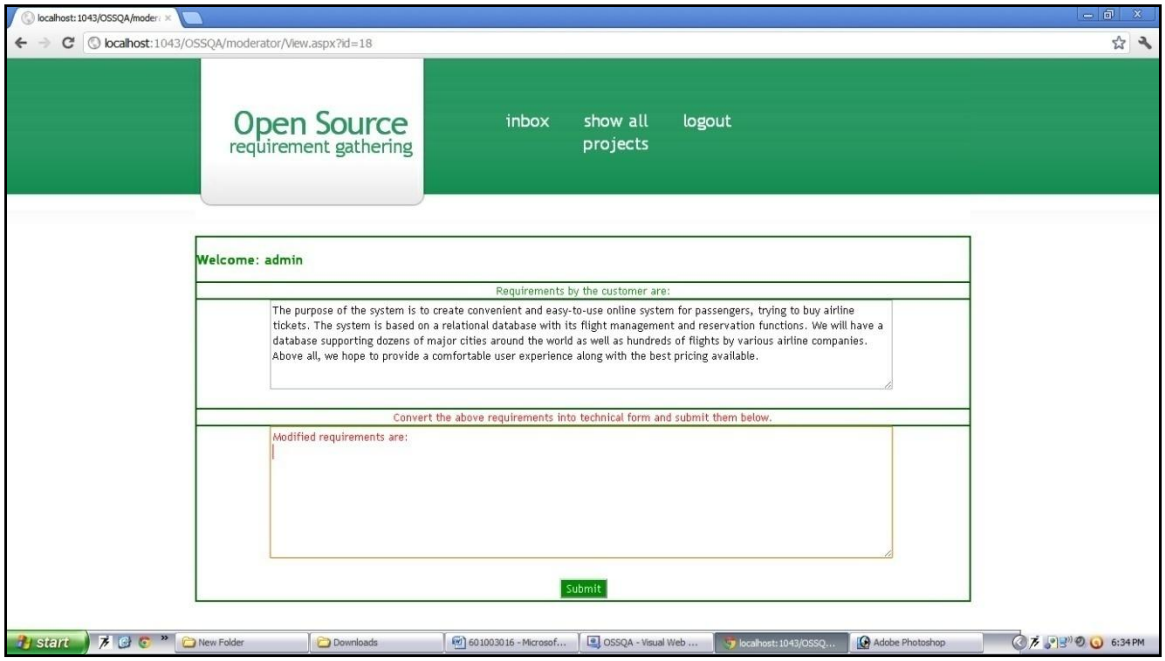

Figure 4.8 Full view of requirement in moderator`s account.

Development of the software-

Once the requirement is converted into technical format it will be visible in the inbox of each and every registered developer on this portal. There is also a search button at the top of developer`s inbox. So that a developer can search the software requirements according to his/her interest.

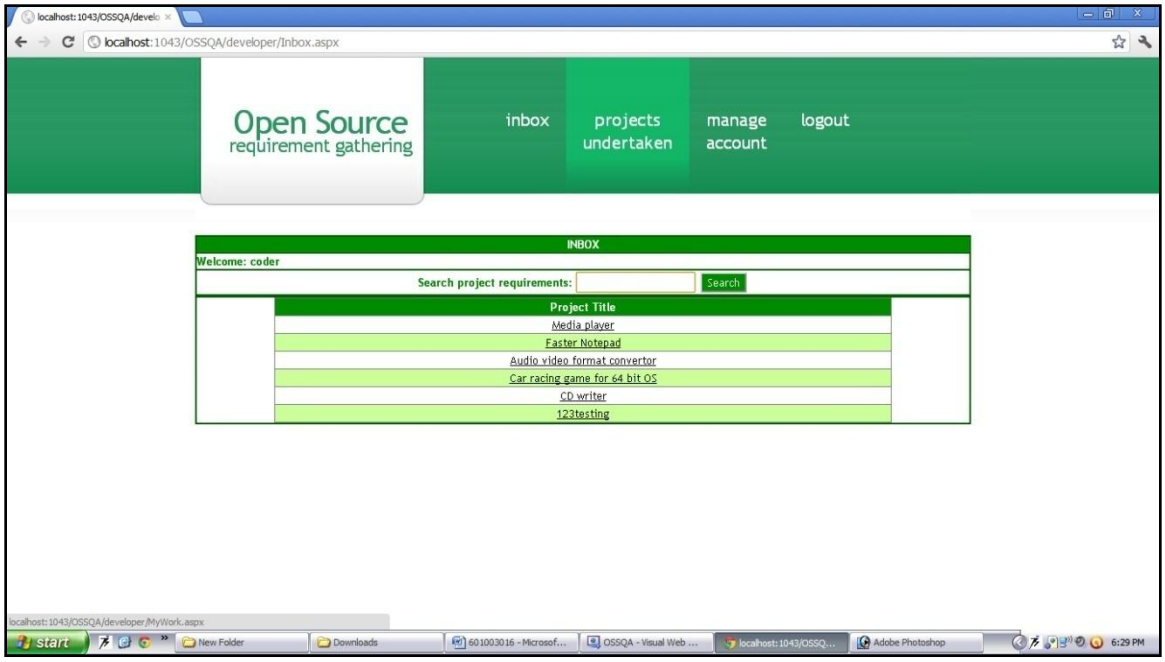

Figure 4.9 Developer`s Inbox.

Developer can view the requirements by clicking the title of the requirements shown in the inbox. If the developer is interested in developing the required software then he/she can click on the 'I will do it' button. It is only in the hand of developer whether he/she wants to develop the required software or not. There is no restriction of any kind on the developers registered on this portal.

When a developer will click on the title of any required software in his/her inbox it will be displayed as shown below:

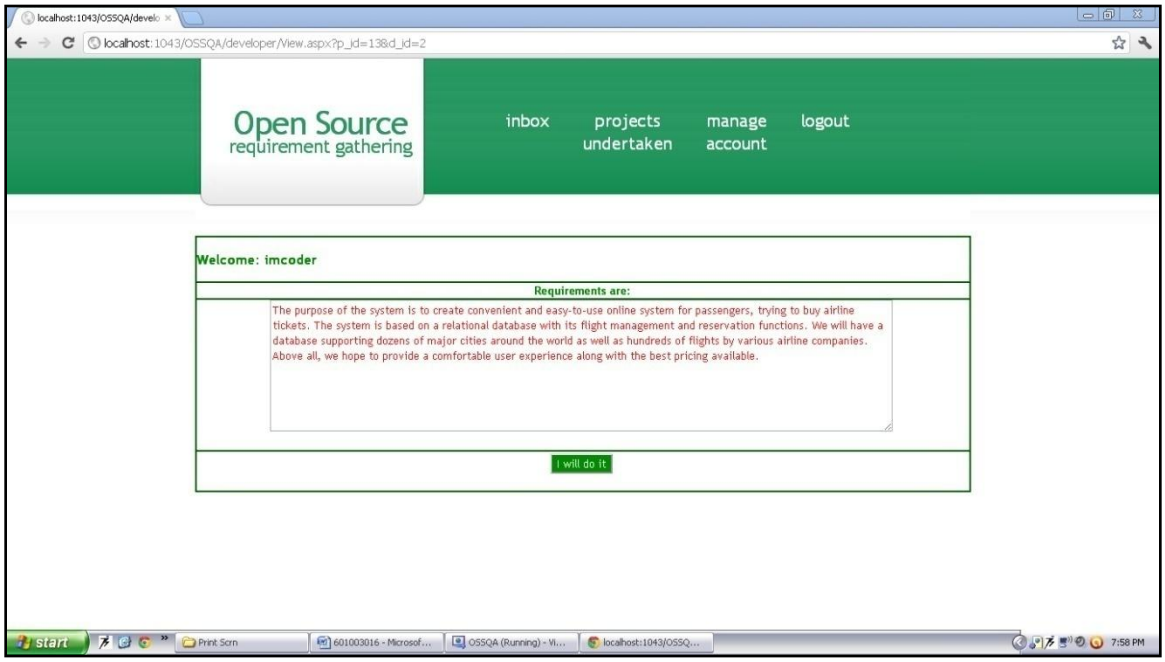

Figure 4.10 Requirements view in developer`s account.

Developer can view the projects undertaken by him/her by clicking ‘projects undertaken’ tab, as shown below:

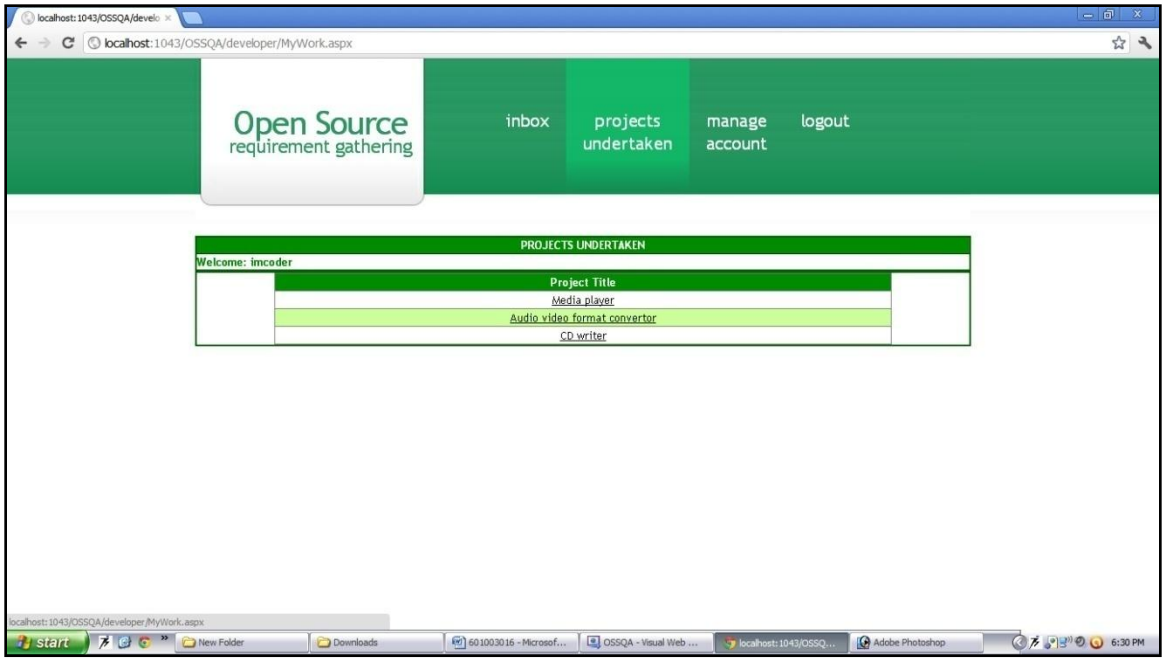

Figure 4.11 Projects undertaken by the developer.

When the developer is ready with the required software he/she can send the download link of the software to the customer. For this he/she will have to type the download link and click the 'Post url' button.

However, the developer can quit the project undertaken by him/her. For this he/she will have to select the project undertaken and click the button 'I want to quit this project'.

Web form for these two activities is shown below:

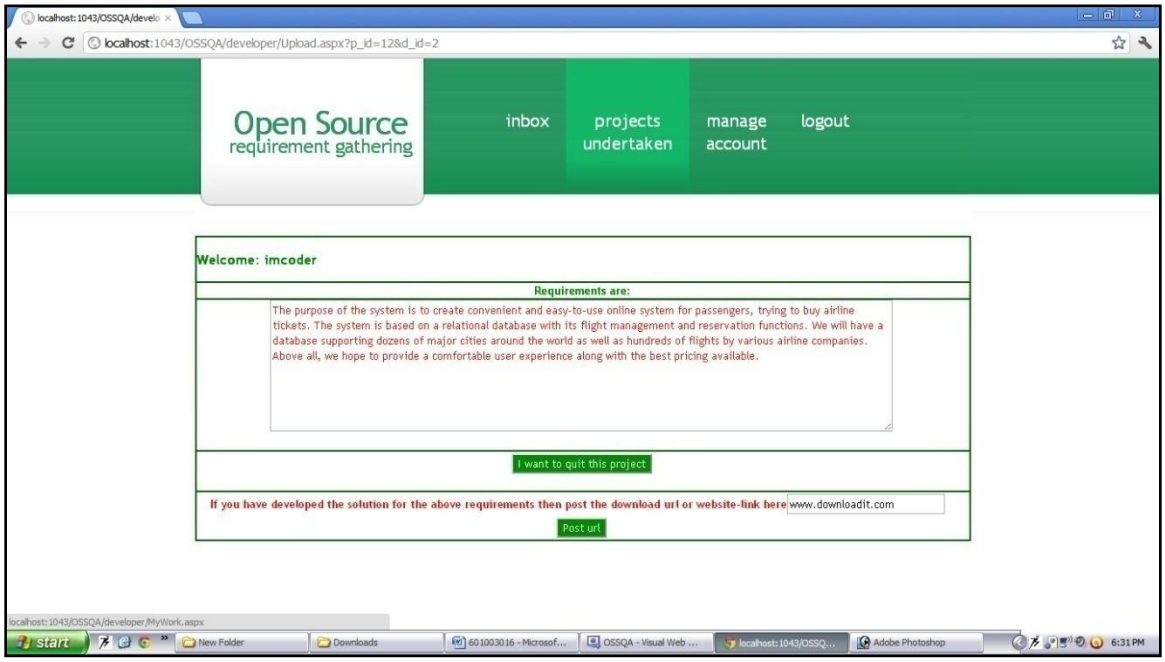

Figure 4.12 Post download link of the required software.

Status of any software-

Four possible status of any software on this portal will be:

a. **Is in queue** – it means that the requirements are not yet converted into technical form by the moderator.
b. **Converted into technical form-** this status shows that the moderator has converted the requirements into technical form.
c. **Developers are working on it**- this status will mention the numbers of developers working on the software.

d. **Developed-** this status will mention the number of developers who have developed the software.

If the required software is in queue then it will be shown as below:

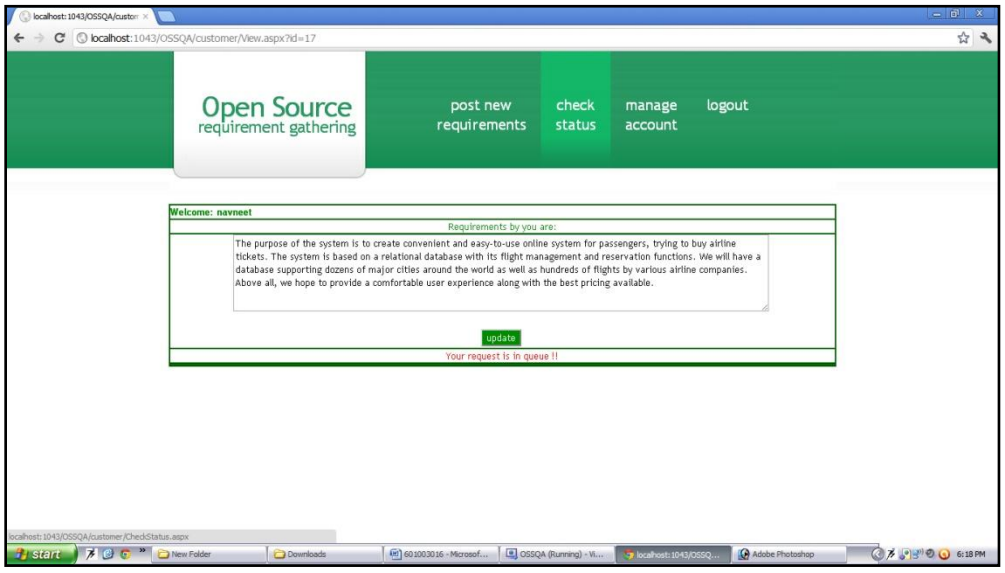

Figure 4.13 Requirement is in queue.

If the requirement is converted into technical form then it will look as shown below:

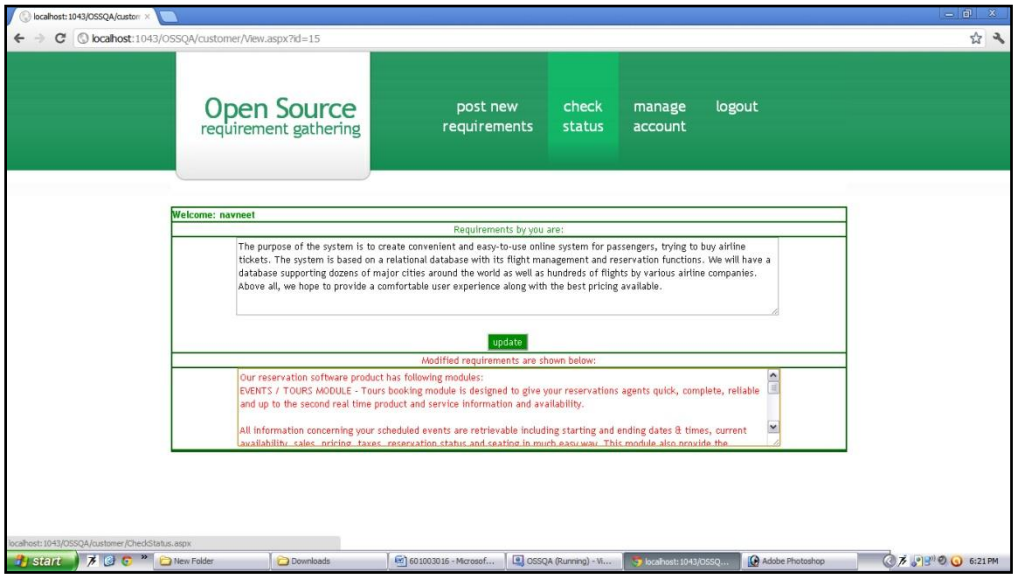

Figure 4.14 Requirements converted into technical format.

If the required software is developed by the developers then the download link(s) of the software will be shown to the customer as shown below.

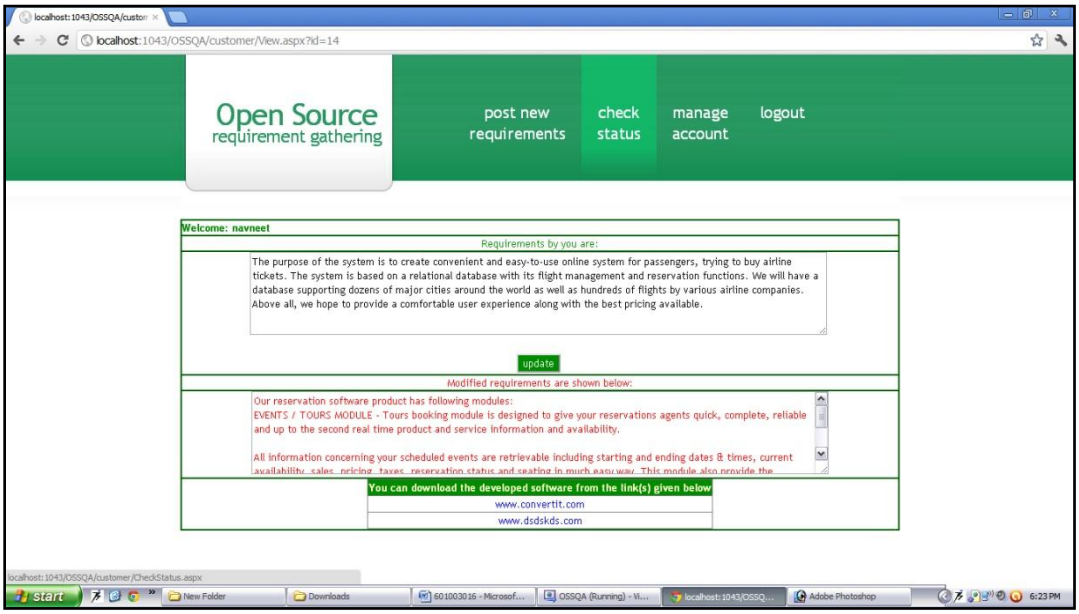

Figure 4.15 Download link of the required software.

If any customer have posted requirements for more than one software than he/she can also view the summary of status of all softwares by clicking the 'check status' tab.

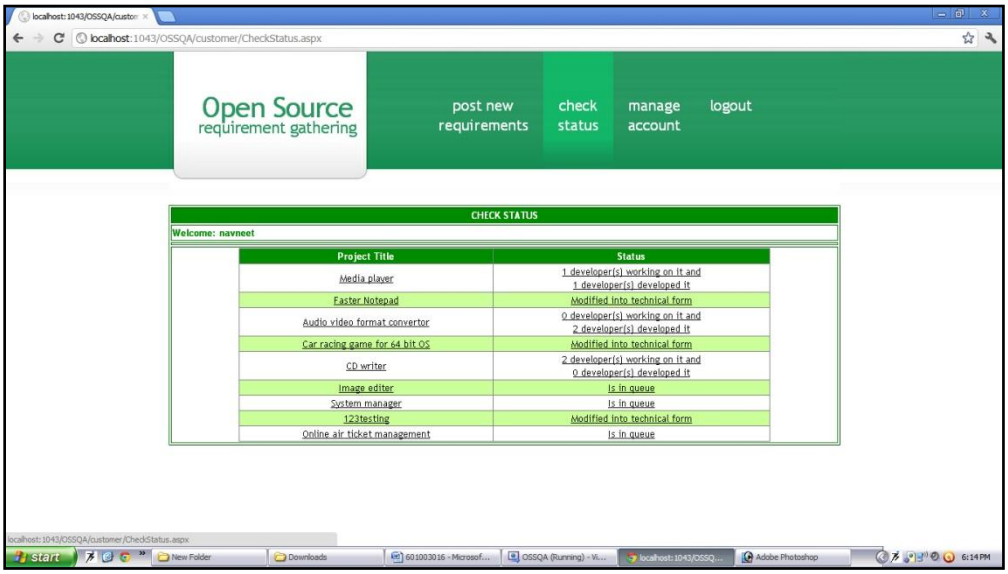

Figure 4.16 Status summary of more than one project.

Similarly the moderator can also view the status summary for all the projects on this portal by clicking the 'show all projects' tab, as shown below:

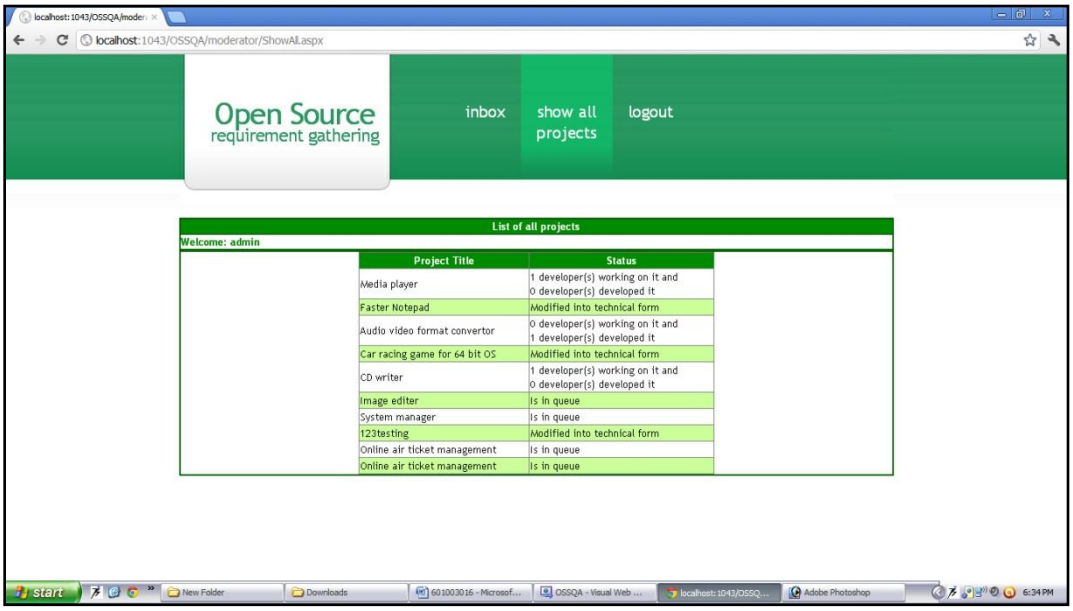

Figure 4.17 Show all projects.

Managing account- Users of this portal can manage their account by clicking the 'manage account' tab, as shown below:

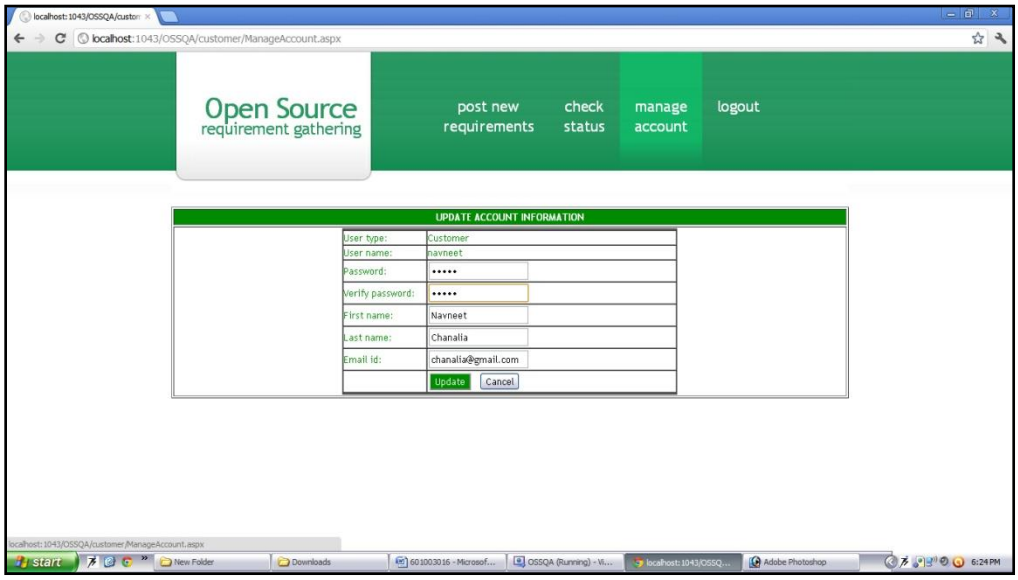

Figure 4.18 Manage account.

Database tables used-

1. *Requirement table* is used to store the requirements submitted by the customer. Each required software is also given a name and unique identity in this table. This table also contains the unique id of the corresponding customer. Modified requirements are also stored in this table.

Table 4.1 Requirement Table

Column Name	Data Type
project_id	numeric(18, 0)
project_name	varchar(50)
project_req_customer	varchar(2000)
project_req_moderator	varchar(2000)
quality_screen_shot	image
customer_id	numeric(18, 0)
moderator_id	numeric(18, 0)
status	varchar(100)

2. *Customer table* stores the information of the customers registered on this portal.

Table 4.2 Customer Table

Column Name	Data Type
customer_id	numeric(18, 0)
mail_id	varchar(50)
uname	varchar(50)
password	varchar(50)
fname	varchar(50)
lname	varchar(50)

3. *Moderator table* stores the information of the moderator.

Table 4.3 Moderator table.

Column Name	Data Type
moderator_id	numeric(18, 0)
uname	varchar(50)
password	varchar(50)
fname	varchar(50)
lname	varchar(50)

4. *Developer table* stores the information of the registered developers.

Table 4.4 Developer table.

Column Name	Data Type
developer_id	numeric(18, 0)
mail_id	varchar(50)
uname	varchar(50)
password	varchar(50)
fname	varchar(50)
lname	varchar(50)

5. *Code table* is used to store the download link of the developed softwares. It also stores the unique id of the corresponding developer and project id.

Table 4.5 Code table.

Column Name	Data Type
project_id	numeric(18, 0)
developer_id	numeric(18, 0)
code	varchar(100)

Chapter 5

CONCLUSION & FUTURE SCOPE

5.1 Conclusion

This thesis encompasses the study of open source softwares and their comparison with the traditional closed source softwares. This thesis also covers the basics of software quality assurance. After studying various papers regarding quality of open source softwares this thesis finds the absence of requirement gathering phase for the open source softwares. Hence, implements a requirement gathering portal, for the open source softwares, keeping in mind the comfort of both developer and end user. This thesis also proposes a development framework for the development of open source softwares.

5.2 Future Scope

As we know that non technical users can not assess the quality of the software using its source code. So in future we can use automated softwares to analyze the quality of the open source software. And the screen shots of these quality results can be posted with the source code of the OSS on the proposed requirement gathering portal. In this way the end user will get his required software along with the proof of quality.

References

[1] C. Gacek and B. Arief, "*The many meanings of open source software*", *IEEE*, vol. 21, pp. 34-40, 2004.

[2] S. Steiniger and G. J. Hay, "*Free and open source geographic information tools for landscape ecology*", Ecological Informatics, vol. 4, pp. 183-195, 2009.

[3] G. Hertel, "*Motivation of software developers in Open Source projects: an Internet-based survey of contributors to the Linux kernel*", *Research Policy*, vol. 32, pp. 1159-1177, 2003.

[4] From Wikipedia. 2010, *Richard Stallman*.

Available: http://en.wikipedia.org/wiki/Richard_Stallman

[5] H. E. Pearson, "*open source licences: open source -- the death of proprietary systems*", Computer Law & Security Review, vol. 16, pp. 151-156, 2000.

[6] S. Raghunathan., "*Open source versus closed source: software quality in monopoly and competitive markets*", Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on, vol. 35, pp. 903-918, 2005.

[7] K. V. Herwig Mannaert, "*The use of open source software platforms by Independent Software Vendors: issues and opportunities*", presented at the International Conference on Software Engineering, Proceedings of the fifth workshop on Open source software engineering 2005.

[8] T. Waring and P. Maddocks, "*Open Source Software implementation in the UK public sector: Evidence from the field and implications for the future*", International Journal of Information Management, vol. 25, pp. 411-428, 2005.

[9] M. Bloch. 2010, "*Open source software in your online business - advantages/disadvantages*", Available: http://www.tamingthebeast.net/articles5/open-sourcesoftware.htm

[10] M. Bloch. 2007, "*disadvantages of open source software*", Available: http://blog.ecomsolutions.net/index.php/2007/12/18/disadvantages-of-open-source-software .

[11] M. Bloch. 2010, "*advantages and disadvantages of open source software*", Available: http://www.softwarecompany.org/advantages-opensource- software.html.

[12] D. M. G. Peter C. Rigby, Margaret-Anne Storey and "*open source software peer review practices: a case study of the apache server*", presented at the International Conference on Software Engineering, Proceedings of the 30th international conference on Software engineering 2008.

[13] A. Beard, H. Kim, "*A survey on open source software licenses*", Journal of Computing Sciences in Colleges, vol. 22 no. 4, April 2007.

[14] Software Engineering, A Practitioner's Approach, Roger Pressman, ISBN 0-07-365578-3 (2001)

[15] Humphrey, Watts S., "*Managing the Software Process*", Addison-Wesley, 1989.

[16] S. Steiniger and G. J. Hay, "*Free and open source geographic information tools for landscape ecology*," Ecological Informatics, vol. 4, pp. 183-195, 2009.

[17] L. Zhao and S. Elbaum, "*A Survey on Quality Related Activities in Open Source*", Software Eng. Notes, vol. 25, no. 3, 2000, pp. 54–57.

[18] A. Mockus, R. Fielding, and J. Herbsleb, "*A Case Study of Open Source Software Development: The Apache Server*", Proc. 22nd Int'l Conf. Software Eng. (ICSE 00), IEEE CS Press, 2000, pp. 263–272.

[19] C. Reis and R. de Mattos Fortes, "*An Overview of the Software Engineering Process and Tools in the Mozilla Project*", Proc. Open Source Software Development Workshop, C. Gacek and B. Arief, eds., Dependability Interdisciplinary Research Collaboration, 2002, pp. 155–175.

[20] D. M. German. GNOME, "*A case of open source global software development*", in Proceedings of the 6th International Workshop on Global Software Development, USA, May 2003.