Requirements Specification for Open Source Software Selection

Ying Yang

University of Tampere
Department of Computer Sciences
Computer Science
M.Sc. thesis
Supervisor: Zheying Zhang
June 2008

University of Tampere
Department of Computer Sciences
Ying Yang: Requirements Specification for Open Source Software Selection
M.Sc. thesis, 64 pages, 13 index and appendix pages
June 2008

Open source software has been widely used. The software world is enjoying the advantages of collaboration and cooperation in software development and use with the advent of open source movement. However, little research is concerned about the practical guidelines of OSS selection. It is hard for an organization to make a decision whether they should use the OSS or not, and to select an appropriate one from a number of OSS candidates. This thesis studies how to select an open source software from requirements engineering perspective. It covers the introduction of open source software and requirements engineering. According to the literature review, a software requirements specification template and a selection model are built. These are further verified in a case study of media player selection.

Key words and terms: open source, open source software, license, requirements engineering, software requirements specification, open source software selection.

Contents

1. Introduction

Open source software (OSS) was a revolutionary concept among computer programmers and users [Kumar, 2007]. It is becoming more popular among developers and users community throughout the world. The software world is enjoying the advantages of collaboration and cooperation in software development and use with the advent of open source (OS) movement [Kumar, 2007]. It gives everyone an opportunity to participate in the development of the software project. Everyone can change the instructions, behaviour and functionality of the OSS programs. A number of OSS programs are built and maintained by a large network of volunteer programmers [Wikipedia-Open source, 2007]. And the Internet enabled OS projects to form and grow [Weber, 2004, p.83]. In practice, a website of an OSS project supports all information for the project, such as documentation, source code and bug databases.

Briefly, OSS is a program whose license gives users the freedom to run it for any purpose, to study and modify the program, and to redistribute copies of either the original or modified program without having to pay royalties to previous developers [Wheeler, 2007]. Well-known examples of OS products include Mozilla Firefox [Firefox, 2008] and Linux [Linux, 2008] operation system. A number of enterprises have OSS projects, such as Nokia [Nokia, 2008], IBM [IBM, 2008]. Meanwhile, OSS becomes option that replaces the commercial proprietary software for enterprises and single users.

There are several advantages of using OSS. First of all, almost all OSS products' price is lower than the proprietary software counterparts'. For example, Comparing with the full version Windows XP Home Edition for which customers have to pay $199, customer can order an Ubuntu/Linux operation system from Internet free of charge. They even do not need to pay the delivery fee. Besides, Computer Economics [Computer Economics, 2008] conducted a survey, which offered respondents a choice of five advantages for OSS. The survey results are shown in Figure 1.

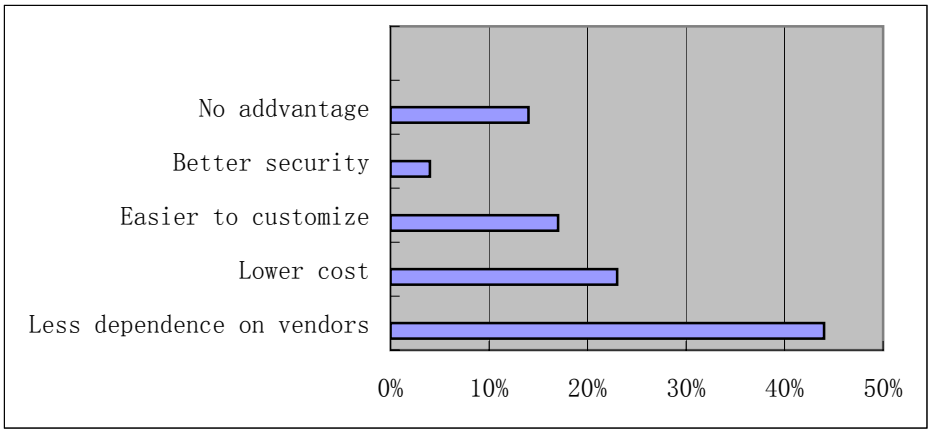


Figure 1. The advantage in use of OSS [Computer Economics, 2008]

According to the survey result, the most important advantage in use of OSS is less dependence on vendors. This indicates that software users want more independence from the vendors. They desire freedom. The OSS can give them the freedom. For example, they do not need to rely on the vendors from whom they bought the OSS products. They can select a proper vendor to supply the maintenance and support servers.

There are also some disadvantages of OSS, especially for small OSS projects. First, some OSS programs have to be complied before using. Second, some OSS programs require the experienced users to use. Third, some OSS programs only run in Linux. Finally, some OSS programs lack document and maintenance support. Useers never like to use the unfriendly and unstable OSS products; even the products are free of charge. In addition, the characteristics of free of charge are not exclusive to OSS. The proprietary software is available in similar ways, such as Visual Studio Express Edition[1].

Although there are some disadvantages, using OSS products could be a good choice for users. European commission sums up the reasons for choosing OSS in various organizations [IDABC, 2007], as shown below.

Political aspect

Issues related to governmental tasks, goals and responsibilities. OSS allows everyone to use, study, modify and distribute, regardless of a person's status, wealth, social status etc [Kumar, 2007].

Economical aspect

A number of studies and user experiences testify various cost reductions when using OSS. The upfront costs of commercial software are not included in the OSS. In general, users get full version of the OSS product, rather than time limited trials [Morgan, 2004].

Social aspect

OSS is a community-based activity. Anybody can join and contribute to the social group.

Technical aspect

OSS is often of a higher quality than its closed or commercial counterparts. Because the source code of OSS products is available, it becomes increasingly easier to study and understand its workings. It forms a big advantage for using OSS.

---

[1] Visual Studio Express Edition: http://www.microsoft.com/express/

Legal aspect

OSS license strictly ensures the users freedom to use, modify, and distribute the programs.

Although the OSS has gotten much attention to the customers, OS was not an idea decreed from the top. The OS movement is a genuine grass roots revolution [O'Reilly, 1999]. Moreover, not all customers have the background knowledge of software development; neither can easily understand the complex license issues. OSS requires a greater degree of computing responsibility than proprietary software [Morgan, 2006]. These restrictions baffle the spread of OSS. It is hard for an organization to make a decision whether they should use the OSS or not, and to select an appropriate one from a number of OSS candidates. To solve this problem, a method for OSS selection is needed. There have been several investigations into the OSS selection. Barbara and Bernd [2006] described an experience in selecting an OS E-Learning tool in their article. They introduced the process of selecting the E-Learning tool. However, they did not explain requirements for OSS selection and how to specify those requirements. Kumar [2007] discussed the factors affecting the OS libraries software selection, installation and maintenance. He did not explain why and how these factors affect the OS libraries selection installation and maintenance. In addition, the factors just affect the OS libraries selection, installation and maintenance. They are not generalized for different OSS selection. Moreover, little research is concerned about the practical guidelines of developing requirements for OSS selection.

OSS and proprietary software have different licenses, different business models, development processes, etc. The requirements engineering (RE) process for OSS selection model is different from the conventional RE for proprietary software development. A key difference is that the information available for OSS is, normally, more than for proprietary software. The OSS has more public information about the software than the proprietary software, such as the software product, the program's source code and so on. Thereby, for OSS selection requirements acquisition can be intertwined with the product evaluation [Barbara and Bernd, 2006]. For example, the users of OSS products can try the products before buying, and there is no time-limited. During the trying, users can evaluate the products and select the one that meets their needs.

1.1. Research Questions

As we have mentioned there are a number of reasons for choosing OSS. Our main concern is how the customers choose a proper OSS product according to their special needs. Therefore, the scope of this thesis is to study how to select an OSS product. In

this study, we adapt the standard software requirements specification (SRS) to describe the customers' needs. The research question addressed in this thesis is the following:

How to specify requirements for OSS selection?

To answer this question, the following subquestions are taken into account:

1. Which factors affect the OSS selection?
2. How are these factors reflected in SRS template?
3. How these factors could be evaluated?

In order to answer these questions, OSS definition, OSS license, RE and SRS will be studied. Through the study, will know factors affecting the OSS selection. With the knowledge a SRS template for OSS selection is created. The template will give insights on the factors' importance. Answers to the second question define the basic elements to create the SRS template. The last question is used for putting the study into practice. It examines how this template works in OSS selection.

1.2. Research Methods

The methods used to conduct this study include: literature review, template and model construction and case study.

1.2.1. Literature

There are plenty of articles written about requirements engineering. Through the RE study, a basic knowledge about requirements and RE process model should be built. OSS is a relatively new field of study. In order to get the criteria, which are applied in the OSS selection, the characteristics of OSS should carefully be studied.

1.2.2. Template and Model

According to the specific characteristics of OSS, this study creates a SRS template for OSS selection. In order to evaluate the SRS template in this study, a model for OSS selection should be built. It introduces the steps for OSS selection and the associated aspects for requirements specification.

1.2.3. Case Study

The media player is basic software in our life. In this thesis, the case study will be performed on media player products. First, we will use the SRS template to create the SRS for Media Player selection. Then, according to the SRS, we get the proper

candidates list and evaluate all of them. Finally, the highest scoring product will be selected. The selection process will follow the selection model. Each criterion will be evaluated. According to the evaluation result, the best media player is selected.

1.3. The Structure of the Thesis

The thesis consists of seven chapters. Chapter 1 provides an overview of his studies by discussing the motivation, the research problems, and research methods. Chapters 2 and 3 introduce the respective background. More specifically, Chapter 2 introduces the basic concepts of OSS, discusses the importance of OSS license in OSS selection, and studies the license compatibility. Chapter 3 presents the concepts of requirements engineering with an emphasis on requirements and the specification. The background discussion leads to studies on requirements specification for OSS selection in the follow-up chapters. Chapter 4 analyses a variety of factors that influence the selection of OSS, and adapts the standard IEEE requirements specification structure for OSS selection. Chapter 5 presents the process model for OSS selection and discusses the use of requirements specification at different stages of the process. Chapter 6 reports a case study, which applies the requirements specification template in the process of selecting an OSS media player for an individual user. Finally, Chapter 7 summarizes the findings and limitations of the thesis work.

2. Open Source Software

What is open source software (OSS)? Although OSS has existed since the 1960's [Weber, 2004], only in the last few years has it been paid much attention. In order to give a thorough explanation of the concept, we introduce a few terms related to OSS. Firstly, what is software? Software is a general term used to describe a collection of computer programs, procedures and documentation that perform some task on a computer system [Wikipedia - Software, 2007]. Simply speaking, it is an integration of programs and related documents. Programs are written in programming languages, such as C, C++ and Java. Most high-level programming languages look like general English. They are relatively easy to understand. The related documents consist of contents, introduction, licenses, and so on.

Secondly, what is source? Here, the source in the context of software development means the source code of software. Software source code composes of functions and directions of a program. The programmer can add or change instructions to adjust the program's behaviour and add functionality [Weber, 2004].

Traditionally, the source code of the software is not available to the users, as it has special value to the software producing companies. Only the producing companies have the right to read and modify the source code. The users also have to get the permission from the vendor if they want to get the source code. Unfortunately, almost all the producers companies never grant the permission to users. Usually, the users need to buy the source code if they want to get it. Sometimes, they can not even get the source code at all. This is the origination of the closed source software. As a result, the software becomes proprietary that is non-free or semi-free. That kind of software is called proprietary or closed source software.

Since the restriction of the source code, users lost the right to study and modify the software. Therefore, the "openness" of source code is a desire. The users want more freedom for the software. They want the permission to read, study and modify the source code. However, it does not mean that software, whose source code is available, is OSS. As an OSS, the users must have the right to modify and redistribute it with or without modification for any purpose and to any person. The Open Source Initiative [1]-(OSI) gives a clear definition. It will be introduced in section 2.2.3.

2.1. History of Open Source Software

Although all the stories related to software are obviously short when comparing them to human civilization, OSS is one of the longest amongst them [Gonzalez-Barahona,

[1] OSI: Open Source Initiative http://www.opensource.org/

2000]. In fact, it could be said that in the beginning of 1950s, there was only free software or OSS. Later on, proprietary software was born, and it quickly dominated the software landscape [Working Group in Libre Software, 2000]. Until recently, the software industry considered OSS as an optional software development approach again.

In early 1950s, when people talked about computer technology, they meant large mainframe computers, which they were often used in military domains. The computer companies supplied the software with the computers. The users did not need to buy the software. And the source code of the software was accessible to the users. The software was widely shared, and programmers collaborated regardless of their employers. The programmers enjoyed learning and tinkering the software [NETC, 2007]. The software products were free of charge, the source code was available, and the programmers could modify the source code. It was very similar to the open source model, but nobody called it, at that time [NETC, 2007], OSS.

In the early 80's, a programmer named Richard Stallman[1] worked for MIT. He once said “Free software is a matter of liberty, not price. To understand the concept, the users should think of free speech, not free beer [Free Software, 2008]." Eventually, he founded the Free Software Foundation[2] (FSF) in 1985 to do just that. Some of the most popular products to come out of the FSF were the GNU[3] suites of free products. His greatest contribution to the computer world is the GNU General Public License (GPL). This license created an avenue and protected people who wanted to create free software [Hart, 2003]. Although FSF had released much free software, it always did not have its own operating system. They began to make it, but its development was slowed down by a number of reasons. And its software was mainly used only in academic circles. By 1990, a GNU OS seemed like it was never going to be released. Linus Torvalds[4], a student at Helsinki University filled this void.

In 1991, Linus got his first computer. He didn’t like MS-DOS operation system, and the other Unix’s systems were all commercial based. That was beyond his means as a student. He wanted an Unix-like operating system that was fun to work on. Since programming was his forte, he began writing his own operating system for fun. This emphasis on fun is a key for understanding why people use and develop OSS. Linus posted his work on Usenet[5] groups and slowly other people started to notice. They looked at the code and started suggesting changes and submitted patches. Linus took these ideas and put them into his project and gave credit where it was due. Later that

---

1 Richard Stallman http://www.stallman.org/

2 Free Software Foundation: http://www.fsf.org/

3 GNU: http://www.gnu.org/

4 Linus Torvalds: http://en.wikipedia.org/wiki/Linus_Torvalds

5 Usenet: Usenet is a worldwide-distributed Internet discussion system.

year, Linus released version 0.1 of the Linux kernel. A kernel is the foundation of an operating system that controls base functions that make the computer work. Programs like the GNU C Compiler and Emacs are the programs, which round out the Linux OS. Linux gained in popularity and by 1994, when version 1.0 was released, it had over 1 million users [Pavlicek, 2000].

In 1998, a group of individuals advocated that the term free software should be replaced by OSS as an expression, which is less ambiguous and more comfortable for the corporate world [Raymond, 1998]. Then, Eric S. Raymond and Bruce Perens [2008] formed the OSI in February 1998. It is a non-profit corporation formed to educate about and advocate for the benefits of OS and to build bridges among different constituencies in the OS community [OSI, 2008]. It gives the intense definition of OS, and has approved a number of OS licenses. All of them are listed on the homepage of OSI.

2.2. The Categories of Software

According to different criteria, software can be divided into different categories. For example, the software can be divided into commercial software and non-commercial software. Commercial software is software being developed by a business, which aims to make money from the use of the software. Although most commercial software is proprietary, “commercial” and “proprietary” is not the same thing. OSS and free software can be commercial software too, and proprietary software can also be non-commercial software. According to the use field or purpose of the software, it can be sorted as follow: [Wikipedia - Software, 2007]:

- Application software: office suites, word processors, spreadsheets, etc.
- System software: operating systems, device drivers, desktop environments, etc.
- Programming tools: assemblers, compilers, linkers, etc.

Figure 2 is quoted from GNU. It shows that most free software is OSS and most OSS falls into the free software category, as most free software and OSS use same license. Twenty-six OSI-approved licenses have been analyzed by the FSF, and only two of these, the Original Artistic License and the Reciprocal Public License, are regarded as non-free licenses [Chen, 2006]. Generally speaking, the licenses of free software are more restrictive than the licenses of OSS. The difference between them will not affect so much on this study; we will not deliberately distinguish them. In fact, they agree more or less on the practical recommendations, and do work together on a number of specific projects, although they have different basic principles [Richard, 2007]. For example, the FSF and the OSI agree on the classification of Free Open Source Software (FOSS) and non-FOSS licenses.

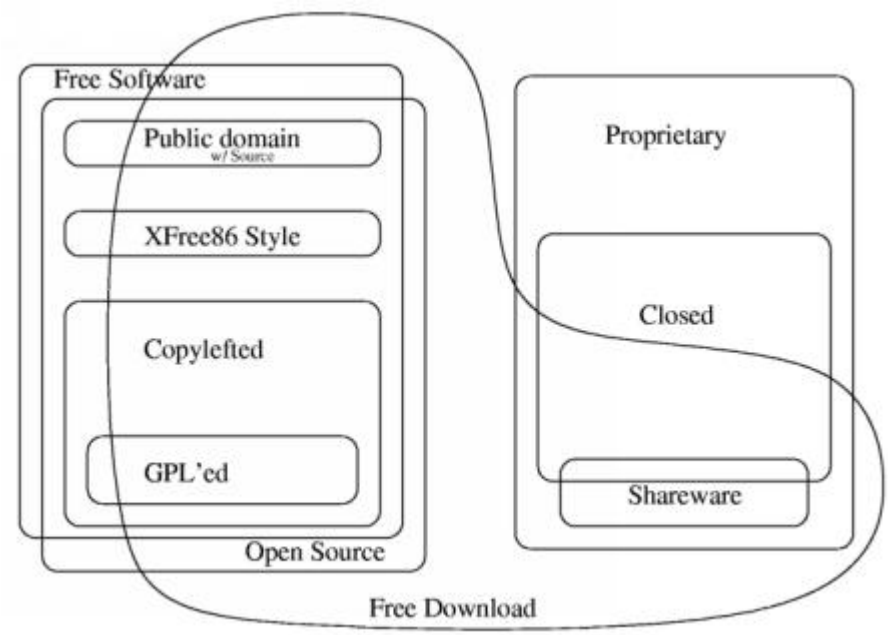


Figure 2. Software Categories [SC, 2007]

In this thesis, the software is sorted into two categories, FOSS and non-FOSS software. The FOSS includes free software and OSS, which can be further divided into Copylefted, GPL'ed, Public domain and Xfree86 Style software. The non-FOSS software includes proprietary software, which can be further divided into Closed software and Shareware.

2.2.1. FOSS

FOSS is an inclusive term generally synonymous with both free software and OSS. It is liberally licensed to grant the right of users to study, change, and improve its design through the availability of its source code [FOSS, 2008].

Free Software

Free software is a matter of freedom, not price. People should be free to use software in all the ways [GNU, 2007a]. The word `free' in free software has a similar meaning as in free speech, free people and free country, and should not be confused with its other meaning associated with zero-cost. It does not mean free of charge, although much free software is free of charge. In particular, free software is software that gives users the freedom to share, study and modify it, either gratis or for a fee. GNU summarizes four kinds of freedom, and explains them precisely [GNU, 2007b]:

- The freedom to run the program, for any purpose .
- The freedom to study how the program works, and adapt it to users' needs. Access to the source code is a precondition for this.
- The freedom to redistribute copies so users can help their neighbour.

- The freedom to improve the program, and release the improvements to the public, so that the whole community benefits. Access to the source code is a precondition for this.

“The freedom to run the program means the freedom for any kind of person or organization to use it on any kind of computer system, for any kind of overall job and purpose, without being required to communicate about it with the developer or any other specific entity. In this freedom, it is the users’ purpose that matters, not the developer's purpose; users are free to run a program for their purposes, and if they distribute it to someone else, she is then free to run it for her purposes, but they are not entitled to impose their purposes on her.”

“The freedom to redistribute copies must include binary or executable forms of the program, as well as source code, for both modified and unmodified versions. (Distributing programs in runnable form is necessary for conveniently installable free operating systems.) It is ok if there is no way to produce a binary or executable form for a certain program (since some languages don't support that feature), but users must have the freedom to redistribute such forms should they find or develop a way to make them.”

“In order for the freedoms to make changes, and to publish improved versions, to be meaningful, users must have access to the source code of the program. Therefore, accessibility of source code is a necessary condition for free software.”

“A program is free software if users have all of these freedoms. Thus, users should be free to redistribute copies, either with or without modifications, either gratis or charging a fee for distribution, to anyone anywhere. Being free to do these things means (among other things) that users do not have to ask or pay for permission.”

Open Source Software

Open Source does not just mean access to the source code. OSI gives an intensive definition of OSS and explanation of it. The distribution terms of OSS must comply with the following criteria [OSS-Definition, 2007]:

Free Redistribution

The license shall not restrict any party from selling or giving away the software as a component of an aggregate software distribution containing programs from several different sources. The license shall not require a royalty or other fee for such sale. By constraining the license to require free redistribution, we eliminate the temptation to throw away a number of long-term gains in order to make a few short-term sales dollars. If we didn't do this, there would be lots of pressure for co-operators to defect.

Source Code

The program must include source code, and must allow distribution in source code as well as compiled form. Where some form of a product is not distributed with source code, there must be a well-publicized means of obtaining the source code for no more than a reasonable reproduction cost preferably, downloading via the Internet without charge. The source code must be the preferred form in which a programmer would modify the program. Deliberately obfuscated source code is not allowed. Intermediate forms such as the output of a pre-processor or translator are not allowed. We require access to un-obfuscated source code because users can't evolve programs without modifying them. Since our purpose is to make evolution easy, we require that modification be made easy.

Derived Works

The license must allow modifications and derived works, and must allow them to be distributed under the same terms as the license of the original software. The mere ability to read source isn't enough to support independent peer review and rapid evolutionary selection. For rapid evolution to happen, people need to be able to experiment with and redistribute modifications.

Integrity of Author's Source Code

The license may restrict source-code from being distributed in modified form *only* if the license allows the distribution of "patch files" with the source code for the purpose of modifying the program at build time. The license must explicitly permit distribution of the software built from modified source code. The license may require derived works to carry a different name or version number from the original software. Encouraging lots of improvement is a good thing, but users have a right to know who is responsible for the software they are using. Authors and maintainers have reciprocal right to know what they're being asked to support and protect their reputations.

No Discrimination Against Persons or Groups

The license must not discriminate against any person or group of persons. In order to get the maximum benefit from the process, the maximum diversity of persons and groups should be equally eligible to contribute to open sources. Therefore we forbid any open-source license from locking anybody out of the process.

Some countries, including the United States, have export restrictions for certain types of software. An OSD-conformant license may warn licenses of applicable restrictions and remind them that they are obliged to obey the law; however, it may not incorporate such restrictions itself.

No Discrimination Against Fields of Endeavor

The license must not restrict anyone from making use of the program in a specific field of endeavour. For example, it may not restrict the program from being used in a business, or from being used for genetic research. The major intention of this clause is to prohibit license traps that prevent open source from being used commercially. We want commercial users to join our community, not feel excluded from it.

Distribution of License

The rights attached to the program must apply to all to whom the program is redistributed without the need for execution of an additional license by those parties. This clause is intended to forbid closing up software by indirect means such as requiring a non-disclosure agreement.

License Must Not Be Specific to a Product

The rights attached to the program must not depend on the program's being part of a particular software distribution. If the program is extracted from that distribution and used or distributed within the terms of the program's license, all parties to whom the program is redistributed should have the same rights as those that are granted in conjunction with the original software distribution. This clause forecloses yet another class of license traps.

License Must Not Restrict Other Software

The license must not place restrictions on other software that is distributed along with the licensed software. For example, the license must not insist that all other programs distributed on the same medium must be open-source software. Distributors of open-source software have the right to make their own choices about their own software.

License Must Be Technology-Neutral

No provision of the license may be predicated on any individual technology or style of interface. This provision is aimed specifically at licenses, which require an explicit gesture of assent in order to establish a contract between licensor and licensee. Provisions mandating so-called "click-wrap" may conflict with important methods of software distribution such as FTP download, CD-ROM anthologies, and web mirroring; such provisions may also hinder code re-use. Conformant licenses must allow for the possibility that **(a)** redistribution of the software will take place over non-Web channels that do not support click-wrapping of the download, and that **(b)** the covered code (or re-used portions of covered code) may run in a non-GUI environment that cannot support popup dialogues.

Copylefted Software

Copylefted software is FOSS software whose distribution terms do not let redistributors add any additional restrictions when they redistribute or modify the software. This means that every copy of the software, even if it has been modified, must be free software [GNU, 2007a].

But non-copylefted free software also exists. Non-copylefted free software comes from the author with permission to redistribute and modify, and also to add additional restrictions to it [GNU, 2007a]. If a program is free but not copylefted, then some copies or modified versions may not be free at all. For example, a software company can compile the program, with or without modifications, and distribute the executable file as a proprietary software product.

GPL'ed Software

GPL'ed software is the one whose license is under GNU GPL. GPL'ed software is copylefted software.

Public Domain Software

Public domain software is software that is not copyrighted. It is a special case of non-copylefted free software, whose source code is in the public domain. The copies or modified versions of public domain software may not be free at all.

Xfree86 Style

The XFree86 Project, Inc is a global volunteer organization, which produces XFree86®**,** the freely redistributable open-source implementation of the X Window System continuously since 1992 [XFree86 2007]. All products of Xfree86 are under Xfree86 license.

XFree86 runs primarily on UNIX® and UNIX-like operating systems like Linux, all of the BSD variants, Sun Solaris both native 32 and 64 bit support, Solaris x86, Mac OS X as well as other platforms like OS/2 and Cygwin.

2.2.2. Non-FOSS

In this study, non-FOSS equals to proprietary software. Proprietary software is software that is not free or semi-free. Its use, redistribution or modification is prohibited, or requires the users to ask for permission, or is restricted so much that the users effectively can't do it freely [GNU, 2007a]. The source code of proprietary software has always kept as a secret. Usually, an individual or a company who developed it owns the source code. Software under this category includes closed software and shareware.

In figure 3, it shows that proprietary software also can be free download. But, the source code is not available. Even if the source code of proprietary software is available, users have no permission to modify and redistribute it, which distinguishs it from the OSS.

Closed Software

Closed software is software whose source code is unavailable.

Shareware

“Shareware is software which comes with permission for people to redistribute copies, but says that anyone who continues to use a copy is *required* to pay a license fee.” Shareware is not free software, or even semi-free. There are two reasons:

1. For most shareware, source code is not available; thus, the users cannot modify the program at all.
2. Shareware does not come with permission to make a copy and install it without paying a license fee, not even for individuals engaging in non-profit activity.

2.2.3. Comparison between OSS and proprietary software

Although some OSS programs are developed by developers' interesting, and free to use, distribute, and modify, it does not mean OSS is anti commercial. Both proprietary software and OSS can be commercial software. The motivation for money (commercial or non-commercial) cannot be a difference between them.

In figure 2, it shows that not all free software and OSS can be freely downloaded and part of the proprietary software can be freely downloaded. The concept “free download” can not distinguish between the free software, OSS and proprietary software.

According to the definitions of OSS and proprietary software, the main difference between OSS and proprietary software is whether the source code of software is available and free to modify, whether the software is free to redistribute. Obviously, the proprietary software does not give the users that permission to read and modify the source code and redistribute the software. Although part of the proprietary software is source code available, and free to download and use, it is not OSS.

Except the difference of the definition, the cost of using OSS and proprietary software is different. Usually, the price of an OSS program is far less than a comparable proprietary program. But, it is the initial price of the software. Here, we will talk about the total cost of ownership (TCO) of them. According to the Northwest Educational

Technology Consortium[1] (NETC), the TCO includes the sale price (initial fee), any hardware and software upgrades, maintenance and technical support, and training (or re-training) [NETC, 2008]. Time and frustration may be hard to measure.

As we have mentioned, the price of an OSS program is usually far less than a comparable proprietary program. Unfortunately it's not clear whether the TCO of OSS is really lower than the comparable proprietary software. Proprietary software companies claim their TCO is lower while OSS companies argue the opposite. One long-term study of Web server deployments found a lower TCO for Linux over Microsoft Windows and Sun Solaris [Orzech, 2002]. But Microsoft alleges lower TCO with its "comprehensive, integrated, easy-to-use stack of technologies" and has its own favourable studies [Cooper, 2003]. The Figure 3 comes from NETC. It shows the opinions both of them.

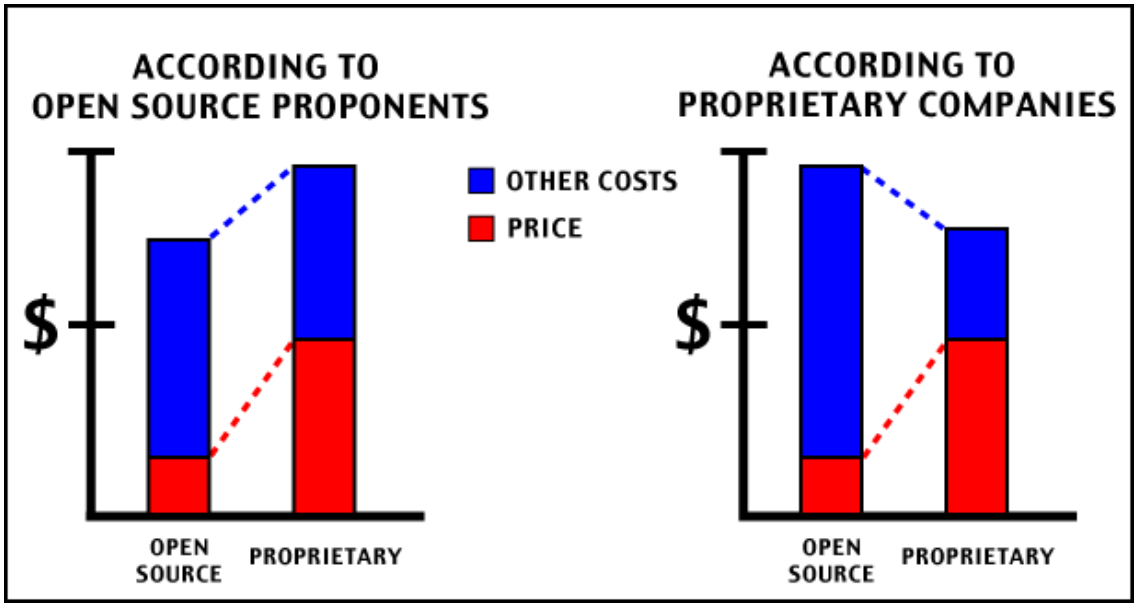


Figure-3 Total Cost of Ownership [TCO, 2008]

Usually, OSS programs may require more skill to deploy and maintain, compared to proprietary programs. For a skilled customer, he/she only needs to pay the price of product. Thus, the TCO of OSS is less than TCO of proprietary software, according to either OS proponents or proprietary companies.

2.3. Open Source Software License

The license under a program is used to define exactly the rights that users have. For example, in most proprietary programs the license withdraws the rights of copying, modification, lending, renting and using in several machines. Usually, licenses under

[1] NETC: Northwest Educational Technology Consortium http://www.netc.org/

proprietary programs specify the users only have the restricted rights to use the program [Gonzalez-Barahona, 2000]. In contrast, the OSS license complies with the terms of the open source definition and is used to specify several goals, like:

- Guaranteeing some basic freedoms of redistribution, modification and use to the users;
- Ensuring some conditions imposed by the authors, for instance, citation of the author in derived works; and
- Guaranteeing that derived works are also OSS.

The authors of software products can choose the different licenses to protect their programs. Most popular and sometimes considered normative OSS licenses are those approved by the OSI based on their Open Source Definition, such as GPL Version 3, LGPL Version 3 and the MIT License. There are a lot of different OSS licences out there, and it can sometimes be a bit confusing if the users are not intimate with the details of each one. Therefore, if the user wants to modify and redistribute his work derived from an OSS program, he has to carefully study the license of the OSS program. Fortunately, although each author could use a different license for his program, the fact is that almost all OSS programs use one of the common licenses, GPL Version 3 [GPLv3, 2007], LGPL Version 3 [LGPLv3, 2007]. Basically, the users just have to study those two licenses. And we will introduce those two licenses and another license MIT in later sections.

2.3.1. GNU GPL and LGPL Licenses

GPL is abbreviation of General Public License. Richard Stallman originally wrote it for the GNU project. Although it was designed for free software, it has approved by OSI. Now, it is a widely used free software and OSS license.

The GPL was carefully designed to promote the production of more free software, and because of that, it explicitly forbids some actions on the software, which could lead to the integration of GPL software in proprietary software. The main characteristics of the GPL are the following [Gonzalez-Barahona, 2000]:

- It allows binary redistribution, but only if source code availability is also guaranteed.
- It allows source redistribution (and enforces it in case of binary distribution).
- It allows modification without restrictions (if the derived work is also covered by GPL).

- Complete integration with other software is only possible if that other software is also covered by GPL.

The GPL allows selling of the copies of software for money, and also allows charging a fee for downloading the software from an Internet site. The fee may be whatever the distributor wishes, however, the source code must be provided. The author also can modify the GPL software to a new software, and then sell it for money, but it must be under the terms of GPL (GPL is copylefted). Therefore, if a company want to make profit by selling the GPL software, it would be quite impossible. Because the users are free to release the software to public. A famous GPL software is the Linux kernel.

The Lesser or Library General Public License (LGPL) is a modified GPL. It is more permissive and less restrictive than GPL. It is especially useful for some software libraries. This makes it possible to combine free software and OSS with proprietary software (GPL does not permit). For example, a project AB is combined with an OSS program A and a proprietary program B. If program A is licensed under the GPL, the derived work program AB is required to be licensed under the GPL. Obviously, it is inconsistent with the license permission of proprietary program B. It is why a number of proprietary software companies call GPL “virus”. However, if A is under LGPL, the derived work AB does not need to be under LGPL. The LGPL encourages greater use of free libraries even in proprietary software projects.

2.3.2. The MIT License

The MIT license is a free software license originating at the Massachusetts Institute of Technology (MIT). It states more explicitly the rights given to the end-user, including the right to use, copy, modify, merge, publish, distribute, sub-license, and/or sell the software. The one condition is that the copyright notice and the permission notice shall be included in all copies or substantial portions of the software.

2.3.3. Copyleft

Copyleft is a play on the word copyright and is the practice of using copyright law to remove restrictions on distributing copies and modified versions of a work for others and requiring that the same freedoms be preserved in modified versions [Wikipedia, 2007e]. GNU gives a formal definition of Copyleft. It is a general method for making a program or other work free, and requiring all modified and extended versions of the program to be free as well [GNU, 2007c]. In general, copyright law allows an author to prohibit others from reproducing, adapting, or distributing copies of the author's work. On the contrary, an author may, through a copyleft licensing scheme, give every person who receives a copy of a work permission to reproduce, adapt or distribute the work as

long as any resulting copies or adaptations are also bound by the same copyleft licensing scheme. It prevents FOSS becoming proprietary.

There are two broad categories of FOSS licences, copyleft licenses and non-copyleft licenses For example, GPL is copyleft license and MIT is not copyleft license. If the users want to modify the product and convert the product into proprietary or require derived works under the same license, they have to know if the license is copyleft or not.

Original work license	Derived product	
	Same license	Proprietary
GPL (Copyleft license)	Yes	No
MIT (Non-copyleft license)	No	Yes

Table. 1 Licenses of the original product and derived products

In Table 1, it shows that if the original product is under coplyleft license, such as GPL, the derived product has to be under the same license, and can not be converted into proprietary product; if the original product is under non-copyleft license, such as MIT, the product need not to be under the same license, and may be converted into proprietary product.

2.3.4. Licenses Compatibility

Different license must have more or less different requirements. If the licenses contain contradictory requirements it impossible to combine source code from such packages in order to create new software packages [O'Riordan, 2006]. For example, if one licence says "modified versions must mention the developers in any advertising materials", and another licence says "modified versions cannot contain additional attribution requirements", then, if someone combined a software package which uses one licence with a software package which uses the other, it would be impossible to distribute the combination because the two requirements cannot be simultaneously fulfilled [Stallman, 2007]. When two OSS projects have incompatible licenses, they can't share code.

There are a large number of OSS licenses, but only a few are widely used. Figure 4 makes it easy to see when common licenses can be combined [Wheeler, 2007]:

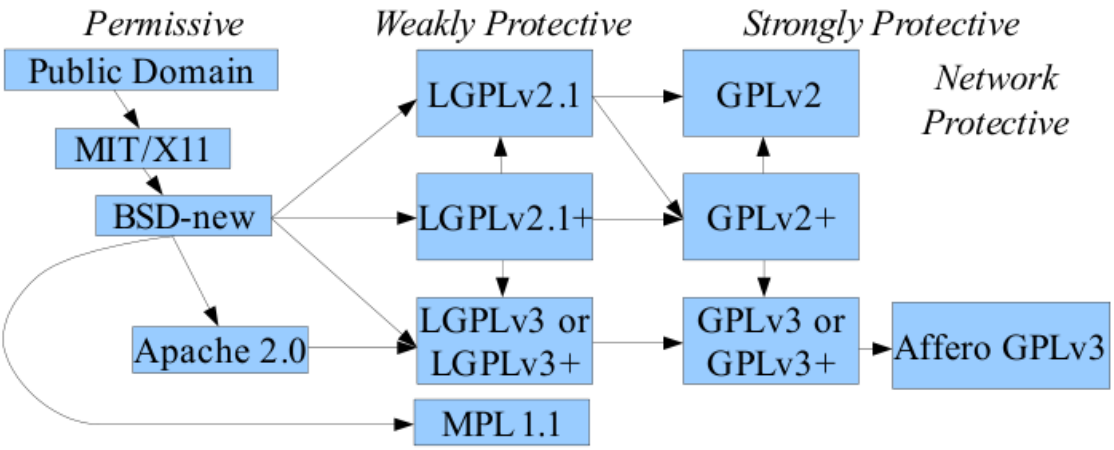


Figure 4 License Compatible [LC, 2007]

In Figure 4, blue rectangles are the names of different OSS licenses. An arrow from box A to box B means that license A is compatible to license B. In other words, users can combine software with these licenses; the combined result has the license of B. To see if software can be combined, just start at their respective licenses, and find a common box the users can reach following the arrows. For example, MIT/X11 licensed software and LGPLv3 licensed software can both reach “GPLv3 or GPLv3+”, they can be combined using GPLv3 or GPLv3+.

Further more, if one product is under GPLv3, the other product is proprietary software, the combination product is under GPLv3. Then, the proprietary software has to convert into OSS. This is why OSS is called “virus”. To solve the problem, the user can use the LGPLv3. LGPLv3 allows the users combine LGPLv3 product with proprietary software, without converting the proprietary software into LGPLv3 software. But any changes to the LGPLv3 product itself must be released under the LGPLv3.

2.3.5. Summary

As we mentioned in previous sections, when selecting the OSS product, users should check whether the license of the product satisfies their needs. When they want to convert an OSS product into a proprietary software product, they should ensure that the license of the product can not be a copyleft license. When they want combine two OSS products, they should check whether the two licenses of the products are compatible or not. If the two licenses are not compatible, users can not share the source code of the two products, and can not combine them.

3. Software Requirements Specification

Every software system can fulfil some purpose. The early stage of a software development project is to understand of the purpose of the system in some detail. This process is requirements engineering. Broadly speaking, RE is the process of discovering that purpose, by identifying stakeholders and their needs, and documenting these in a form that is amenable to analysis, communication, and subsequent implementation. In details, requirements engineering can be characterized as an iterative process of discovery and analysis, designed to produce an agreed-upon set of clear, complete, and consistent system requirements [Robinson and Pawlowski, 1999].

The RE of a software project is vital to its success. It sets the criteria to evaluate whether the software meets the purpose for which it was intended. All follow-up development activities are influenced or driven by it. RE is, arguably, the most important activity performed during the development of software intensive systems. As the bad requirements, a number of systems have been delivered late and over budget [Kotonya and Sommerville, 1998]. Deficient requirements are the single biggest cause of software projects failures. From studying several hundred organizations, Capers Jones [1996] discovered that RE is deficient in more than 75% of all enterprises. Furthermore, the cost of repairing requirements-related problems dramatically increases as the software development process progresses [Boehm and Papaccio, 1988]. It is therefore evident that, getting requirements right might be the single most important and difficult part of a software project.

Before introducing the requirements engineering process, we have to know what are requirements. Requirements are a specification of what should be implemented. They are descriptions of how the system should behave, or of a system property or attribute. They may be a constraint on the development process of the system [Sommerville and Sawyer, 1997].

Requirements include a set of functional requirements and non-functional requirements. The functional requirements describe the behaviour of the system, such as what the users want a system to do, which can be modelled with use case [Zhang, 2007b]. Sometimes, functional requirements are called behavioural or operational requirements. They specify the inputs to the system and the outputs from the system and the behavioural relationships between them. The important point to note is that **WHAT** is wanted is specified, and not **HOW** it will be delivered. The non-functional requirements are the ones, which impose constraints on the design or implementation, such as performance engineering requirements, quality standards, or design constraints.

Meanwhile, the requirements can be sorted according to the abstraction level. There are four levels, business requirements, user requiements, system requirments and software requiements.

Business requirements

Business requirements are derived from business goals or objectives. They are the essential activities of an enterprise and the reason for developing systems and software in the first place. If the system does not support the business requirements effectively and efficiently, they have no reason for being -- Businesses exist to make money [Kasse Initiatives, 2004]. The business requirements should articulate how the product's developers and their customers will benefit from this product, and what organization or customer request the system. Business requirements include vision and scope document. A short vision statement describes what the product could ultimately become. The scope description should summarize the major features included in the initial release and subsequent releases.

User requirements

User requirements are written for customers in natural language with diagrams. They provide the services and the operational constraints, for example, user case or scenario descriptions.

System requirements

System requirements are the detailed descriptions of the system services. For example, testing, quality assurance, project management. And the schedule and budget can be estimated in system requirements documents.

Software requirements

Software requirements are written for developers. They provide detailed software description, which can serve as a basis for a design or implementation.

3.1. Requirements Engineering

Different organizations tackle RE in radically different ways [Kotonya and Sommerville, 1998]. Although there is little uniformity in authors' terminology or decomposition, a number of RE process phases/activities have been proposed. A process model is a simplified description of the RE process. But, no single model offers a complete understanding of the process. When describing processes in detail it is usual to produce several different types of models giving different process information. At an abstract level, the Linear RE Process Model shown in Figure 5 can describe most requirements engineering processes.

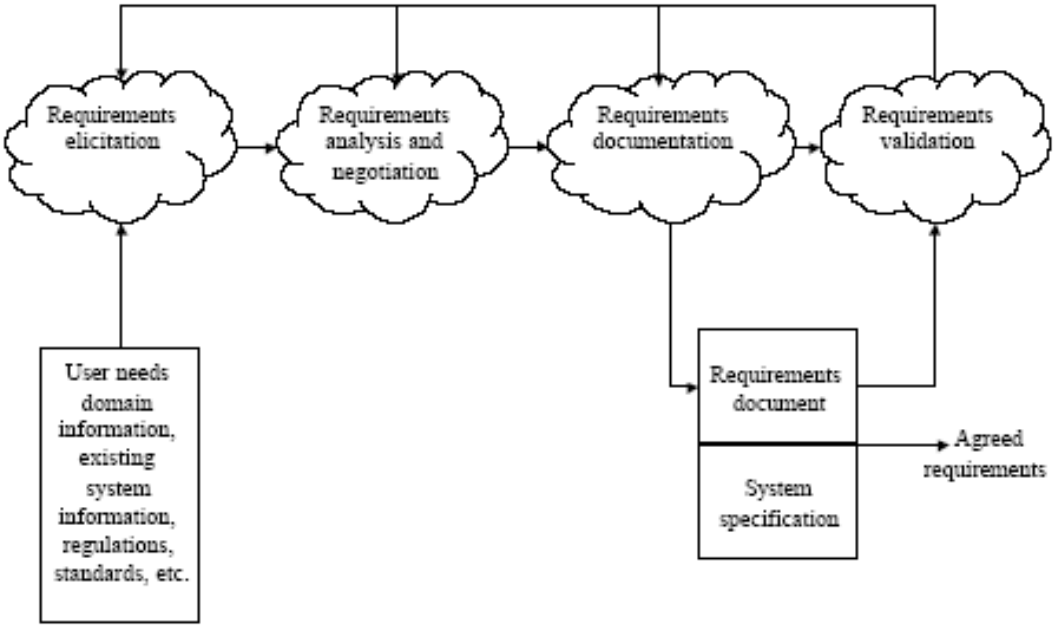


Figure 5. Linear RE Process Model [Kotonya and Sommerville, 1998, p32]

Kotonya and Sommerville [1998, p32] describe the activities in the RE process as follows:

Requirements elicitation

The system requirements are discovered through consultation with stakeholders, from system documents, domain knowledge and market studies. Other names for this process are requirements acquisition or requirements discovery.

Requirement analysis and negotiation

The requirements are analyzed in detail and different stakeholders negotiate to decide which requirements are to be accepted. This process is necessary because there are inevitably conflicts between the requirements from different sources, information may be incomplete or the requirements expressed may be incompatible with the budget available to develop the system. There is usually some flexibility in requirements, and negotiation is necessary to decide on the set of agreed requirements for the system.

Requirements documentation

The agreed requirements are documented at an appropriate level of detail. In general, there needs to be a requirements document, which is understandable by all system stakeholders. This usually means that the requirements must be documented using natural language and diagrams. More detailed system documentation such as system models may also be produced.

Requirements validation

There should be a careful check of the requirements for consistency and completeness. This process is intended to detect problems in the requirements document before it is used as a basis for system development.

In this study, we focus on the activity of requirements documentation (requirements specification). We will create a SRS template for OSS selection.

3.2. Software Requirements Specification

SRS is a document that contains statements of requirements. In different organizations, this document may have different names, such as Requirements document, Functional specification or SRS. In this study, we use SRS to represent the document that specifies requirements.

In software development, SRS provides a representation of the software for the customer's review and approval. It sets out what the software should do without specifying how it should be done. According to Kotonya and Sommerville [1998, p15] it describes the following:

1. The services and functions, which the system should provide.
2. The constraints under which the system must operate.
3. Overall properties of the system, i.e. constraints on the system's emergent properties.
4. Definitions of other systems, which the system must integrate with.
5. Information about the application domain of the system, e.g. how to carry out particular types of computation.
6. Constraints on the process used to develop the system.

There are some SRS templates; for example, the IEEE/ANSI 830-1998 standard Requirements document structure [IEEE Std 830, 1998]. It is available in Appendix A. The IEEE standard structure contains 6 chapters, introduction, overall description, external interface requirements, communications interfaces, other non-functional requirements and other requirements. Chapter 1 introduction contains the purpose, conventions, intended audience and reference of the SRS, and the scope of the product. Chapter 2 overall description describes the general factors that affect the product and its requirements. This chapter does not state specific requirements. Instead, it provides a background for those requirements, which are defined in detail in Chapter 3, Chapter 4, Chapter 5 and Chapter 6 of the SRS. Those chapters include external interface requirements, communications interfaces, other non-functional requirements and other requirements of the SRS. They should contain all of the software requirements to a

level of detail sufficient to enable designers to design a system to satisfy those requirements, and testers to test that the system satisfies those requirements.

The IEEE standard structure is a generic standard, which is intended apply to a wide range of requirements documents. In general, not all parts of the standard are required for all requirements documents. Each organization should adapt the standard depending on the type of systems it develops.

3.3. Characteristics of A Good Software Requirements Specification

Requirements play a driving role during product creation. The requirements are captured in a requirements specification. A good SRS clearly communicates to the stakeholders that the software features described in the document meet the needs of the business [Software Requirements Inc, 2008]. Stakeholders form the requirements source and they include a wider range more than human beings, but also others such as the (physical) environments [Zhang, 2007a].

A SRS should be clear, concise, consistent and unambiguous. It's important to note that an SRS contains functional and non-functional requirements only; it doesn't offer design suggestions, possible solutions to technology or business issues, or any other information other than what the development team understands the customer's system requirements to be. A well-designed SRS has the following characteristics [Donn Le Vie, 2007]:

Complete

SRS defines precisely all the go-live situations that will be encountered and the system's capability to successfully address them.

Consistent

SRS capability functions and performance levels are compatible, and the required quality features (security, reliability, etc.) do not negate those capability functions. For example, the only electric hedge trimmer that is safe is one that is stored in a box and not connected to any electrical cords or outlets.

Accurate

RS precisely defines the system's capability in a real-world environment, as well as how it interfaces and interacts with it. This aspect of requirements is a significant problem area for a number of SRSs.

Modifiable

The logical, hierarchical structure of the SRS should facilitate any necessary modifications (grouping related issues together and separating them from unrelated issues makes the SRS easier to modify).

Ranked

Individual requirements of an SRS are hierarchically arranged according to stability, security, perceived ease/difficulty of implementation, or other parameter that helps in the design of that and subsequent documents.

Testable

An SRS must be stated in such a manner that unambiguous assessment criteria (pass/fail or some quantitative measure) can be derived from the SRS itself.

Traceable

Each requirement in an SRS must be uniquely identified to a source (use case, government requirement, industry standard, etc.)

Unambiguous

SRS must contain requirements statements that can be interpreted in one way only. This is another area that creates significant problems for SRS development because of the use of natural language.

Valid

A valid SRS is one in which all parties and project participants can understand, analyze, accept, or approve it. This is one of the main reasons SRSs are written using natural language.

Verifiable

A verifiable SRS is consistent from one level of abstraction to another. Most attributes of a specification are subjective and a conclusive assessment of quality requires a technical review by domain experts. Using indicators of strength and weakness provide some evidence that preferred attributes are or are not present.

4. Software Requirements Specification for OSS Selection

In this study, the SRS is used for OSS selection rather than software development. The goals, users, abstraction levels are different from a typical software requirements specification.

In software development, SRS is used for providing a representation of the software for the customer's review and approval, and communicating the requirements to customers, engineers and managers. SRS for OSS selection is for selecting an OSS product from all candidates. It forms criteria for evaluating all candidates. It shows WHAT will be evaluated, not HOW will be evaluated.

In software development, SRS is used for two parties. Party A: The person who will buy or use the software, such as software customer and user. Party B: The person who will sell or develop the software, such as software sale staff, project managers, software engineers, software test engineers, software maintenance and support staff, document writer and training personnel. SRS for OSS selection is used for the person who will buy or use the software. It is only one party, i.e. customers and users.

In software development, SRS is designed for a certain product. It explains what the product will do, where the product will work and how well the product will run. Besides the high-level business requirements and user requirements, an important component of SRS is the detailed functional software requirements and non-functional requirements. The detailed software requirements provide developers important information of what shall be implemented, and how the implementation behaves. SRS for OSS selection is designed for an uncertain product. It is used to evaluate what and how all candidates' products have already done. The specification is therefore not specific for application developers, and can remain at a relatively high abstraction level, i.e. user requirements might be detailed enough for OSS selection.

Moreover, the OSS selection specification may include components different from conventional SRS. For example, conventional SRS does not conclude the cost of the product (usually, it is in project plan). However, the cost is an important factor affecting the selection. In SRS for OSS selection, it belongs to business requirements. However, the IEEE standard Requirements document structure is still useful. We will modify it according to our needs. Next, we will analyse the factors, which affect the OSS selection.

4.1. Factors Affecting OSS Selection

There are a number of factors that affect the OSS selection. As we mentioned in Chapter 1, Kumar [2007] gave factors, which affect the OS libraries software selection, installation and maintenance in his article, such as OSS licenses, functional modules, stable releases, developers and users community, user interface and documentation. However, those are specially used for OS libraries selection. In this study, we analyze

the factors affecting a generic OSS products selection. According to Metcalfe [2008] and IDABC [2007], we describe the following factors by alpha sort, cost, community, development group, documentation, functionality, license, lifecycle, market share, security, supporting service and usability, and explain why these factors are important for OSS selection.

Community

One of the most important aspects of OS is the community [Golden, 2005, p.21]. It does most of the testing and provides quality feedback. Unlike the proprietary software, which uses financial resources to test the software, the community is the resource for OSS testing.

The community includes user community and developer community. The user community consists of the people who use the software and participate in some way, such as reporting bugs or other problem of the software, giving feedback on functionality that the software does not work well or the user need. The developers can modify and improve the software according to user requirements. There is no explicit line between the user community and the developer community, and the role of the community member can change from a user to a developer, and vice versa.

A large and active community often indicates the acceptance of the software and its quality. If the software was not good enough to use, there would not be so a number of people who cared about its development [Duijnhouwer and Widdows, 2003].

The community can give the free support to the users. The user can ask question and find answers in community by using mailing list or forums. It is especially important for an individual user, if he/she wants to seek free support from the community.

Cost

Obviously, cost is an important issue for every user. In this study, the cost is total cost of ownership. It includes the purchase cost and any other cost related to the product. Sometimes, the installation of large OSS is hard for customers, especially for the users who do not have an experience of similar software. Therefore, they may need the help. In general, the cost of installing a large OSS is more than a normal proprietary software. For example, large Linux installations need between 25 percent and 40 percent more full-time support resources than Windows or commercial Unix systems. Making financial matters worse, high-demand Linux experts can charge premiums as high as 40 percent more than what their Windows and Unix competitors charge [Joch, 2004].

Usually, most OSS products do not cost anything to get, or have a relatively nominal acquisition cost (e.g. a fee for a boxed CD-ROM set or delivery fee). The purchase cost may be the total cost for a single user. However, for enterprise users, they

have to consider the total cost of ownership, which includes the purchase cost, and ongoing cost for support, such as installation cost, training cost, and maintenance cost.

Development group

The development group is also important for the OSS. The skilled and experienced developers are important factors for OSS quality. And the developers' motivation is another affecting the OSS quality. The motivation behind a certain project can explain the rationale for intended use and how serious the developers are about the project [Golden, 2005].

Documentation

OSS documentation is often lagging behind the status of the application, since especially user documentation is often written only after functionality is created [Scacchi, 2002]. User documentation, including tutorial and frequently asked question (FAQ), explains how to install and to use the software. In addition, the developer documentation is another important OSS documentation. It contains developing information. It provides the insight into the OSS. If the user wants to modify the source code, the developer documentation can explain what a section of code does, how to use and change it and why it works like it does. The user also can find the history of bug fixes, feature changes of the product, etc. When we select OSS product, we should check if the project's site of the product has the documentation.

Functionality

The most basic reason why the customers select the OSS is that the software does what they want it to do. The functional requirements show capabilities of the software. Therefore, when users select software, they have to check, if the software provides the needed functions or services. Usually, a number of OSS products have a brief description of the capabilities. However, there is not ideal software, which provides all users' expecting functionalities. It often needs to add the missing functionalities. OSS has a unique option; the customers can add the missing functionalities by modifying the source code. Thus, when the customers want to add the missing functionalities, they should consider the possibility and cost.

License

As mentioned before, OSS gives the users more freedom than proprietary software. The freedom comes from the OSS licenses. The OSS licenses give the users permission to modify, redistribute the software. However, different developers may use different licenses, which are determined by their motivations. The users have to pay attention to licenses issues, when they select the OSS products. For individual users who use OSS

products without any further modification and redistribution, they do not need to care about the license issues much. For users who will combine, modify and redistribute the OSS applications, especially for the ones who convert the OSS product into a proprietary product, the license issues shall be taken into account carefully. For example, the copyleft'ed software can not be converted into proprietary software, and if the licenses of two products are not compatible, the two products can not be share the source code each other.

OSS Lifecycle

A status of an OSS in its lifecycle indicates a product's stability. A product at its early development stage is usually full of bugs [Golden, 2005, p.103]. The older a project, the less likely the developers will suddenly stop [Duijnhouwer and Widdows, 2003]. It, however, does not mean the older the better. It also depends on the release status. Whether the software is released regularly and whether the new version uses the new technologies and method.

It is often considered impossible to write completely bug-free software of any real complexity [Software Bug, 2008]. Successful software needs to fix the bugs and release the new version. Typically a product needs to reach its 1.0 release prior to being considered for enterprise use. This is not to say that a number of pre-1.0 versions of software are not very good indeed, e.g. Mozilla's 0.8 release of its FireFox browser was polished and mature [Metcalfe, 2008]. For enterprise users, they need the most recent stable release of the software. They also can fix the bugs by themselves, since the source code of the software is available. But, they have to consider the cost, time and possibility.

Market Share

If the users get the OSS from the website like SourceForge[1], which provides the statistics of the each product, the users should know how a number of times it has been downloaded. Sometimes, it is important to know how popular the OSS is. Most successful OSS should have a number of users. A product with a large installed base (Apache for example) provides additional stimulus to form communities. For popular OSS product, there are a number of third parties can provide the support.

Security

OSS gives both attackers and defenders great power over system security [Cowan, 2003]. The characteristic of "source available" is a double blade. It makes the source code available to attackers too. OSS is neither more nor less secure than closed source

[1] SourceForge: http://sourceforge.net/index.php

software. There are famously secure examples of both OSS and closed source software just as there are infamously insecure examples of both open source software and closed source software [Yeates, 2005]. The user has to balance the security requirements with other requirements, when he/she selects OSS product.

Supporting service

There are two types of support, free support and paid support. Usually, the community give the free support. User can ask question and give feedback through the community. The common ways are mailing list and forum. When a user selects an OSS product, he/she should consider how active the community is?

Besides community support, professional support can also be purchased. Unlike the conventional proprietary software, the users have to choose the support from the vendors of the products; the OSS user may have several choices. Paid support is available from a diversity of companies, ranging from large corporations such as IBM and Sun Microsystems, to specialist OS organizations such as Red Hat and MySQL, to local firms and independent contractors [Metcalfe, 2008].

Usability

Open source communities have successfully developed a great deal of software although most computer users only use proprietary applications. The usability of OSS is often regarded as one reason for this limited distribution. [Nichols and Twidale, 2002]. The usability includes how easy the OSS is to install and use. A highly usability program is easy to learn and use. Generally, a graphical user interface (GUI) is friendly interface. However, the usability depends on who are the end users. For example, for some senior software developers they may appropriate the command line interface for certain tasks.

These factors form resources of requirements and are comprised as an important part of SRS template for OSS selection. In this thesis, the SRS template includes three types of requirements: Business requirements, Functional requirements and Non-functional requirements. Business requirements are derived from business goals or objectives. They are the essential activities of an enterprise and the reason for developing systems and software in the first place [Kasse Initiatives, 2004]. They form the reason for selecting systems and software in first place, and include cost and market share of the product. Traditionally, the functional requirements describe the functionalities, which the software will be implemented. In this study, however the functional requirements describe functionalities available to meet users' needs. According to the functional requirements, users can evaluate an existing candidate, and know what functionalities are ready-made, what functionalities are missing, what functionalities are wanted and what functionalities are un-wanted. Non-functional

requirements are requirements, which specify criteria that can be used to judge the operation of software, rather than specific behaviours [Non-functional requirements, 2008]. In this study, the non-functional requirements contain community, developers group, documentation, license, security, support, usability and version.

4.2. Requirements specification template for OSS selection

The purpose of the SRS is to select a software application rather than to implement it. Accordingly, we will adapt the IEEE SRS structure to meet our purpose. As mentioned in section 3.2, conventional SRS is to reduce the gap between the customers and developers. It lets the customers know what are the features they really want, and let the developers know what they will develop. However, the SRS for OSS selection is designed for the customers, and help them to clarify what they want.

The SRS for OSS selection describes the general factors that affect the product selection. It does not state specific software requirements for the designers. The specific software requirements are designed for designers to design a system to satisfy those requirements, and testers to test that the system satisfies those requirements. Instead, it provides the general user requirements, and makes them easier to understand in OSS selection.

The SRS template includes four parts, Introduction, General description, Requirements and Appendix. The following is the structure of the SRS template. The details can be found in Appendix B.

1. Introduction
   - 1.1. Purpose
   - 1.2. Scope
   - 1.3. Definitions, acronyms and abbreviations
   - 1.4. References
2. General Description
   - 2.1. Product Features
   - 2.2. Operating Environment
   - 2.3. User Characteristics
3. User Requirements
   - 3.1. Functional Requirements
   - 3.2. Non-functional Requirements
4. Appendix

Chapter 1 is the introduction of the SRS and product. It introduces the purpose of the SRS, the scope of the product, the definitions, acronyms, and abbreviations and the references of the SRS. According to the study of Chapter 1, users can select the OSS

candidates from a number of sources. Chapter 2 presents the general description of the product. It explains what main features the product has, where the product run and the users characteristics of the product. According to the general description of the product, users can do the pre-selection, and eliminate the unqualified candidates. Chapter 3 gives the user requirements. It presents the detailed functional requirements and non-functional requirements of the product. Those requirements form the criteria for evaluating the remained candidates. The last chapter of the SRS is Appendix. It makes the SRS easier to use.

5. Open Source Software Selection Model

SRS describes WHAT should be taken into account when selecting an OSS. In this chapter, we further discuss how the selection process is performed, and how the factors mentioned in the prior chapter can be evaluated in practice.

The basic steps for evaluating all programs, both OSS and proprietary, are essentially the same [Wheeler, 2007]. Both processes include identifying the candidates, comparing the candidates, analysing the top candidates and making the decision. However, the details in each step are different. A key difference is that the available information for OSS is usually more than for proprietary software. For example, source code, analysis by others of the program design, discussion between users and developers on how well it is working, and so on [Wheeler, 2007]. Therefore, when the customers select an OSS, they should consider all information related to the OSS product. For example, the following criteria can be adopted for OSS libraries selection; functionality, cost, developer and user community, user interface, stable releases, Documentation and so on [Kumar 2007].

In this thesis, the selection and evaluation of OSS is from the RE viewpoint. The requirements form the selection criteria. Figure 6 shows the process model.

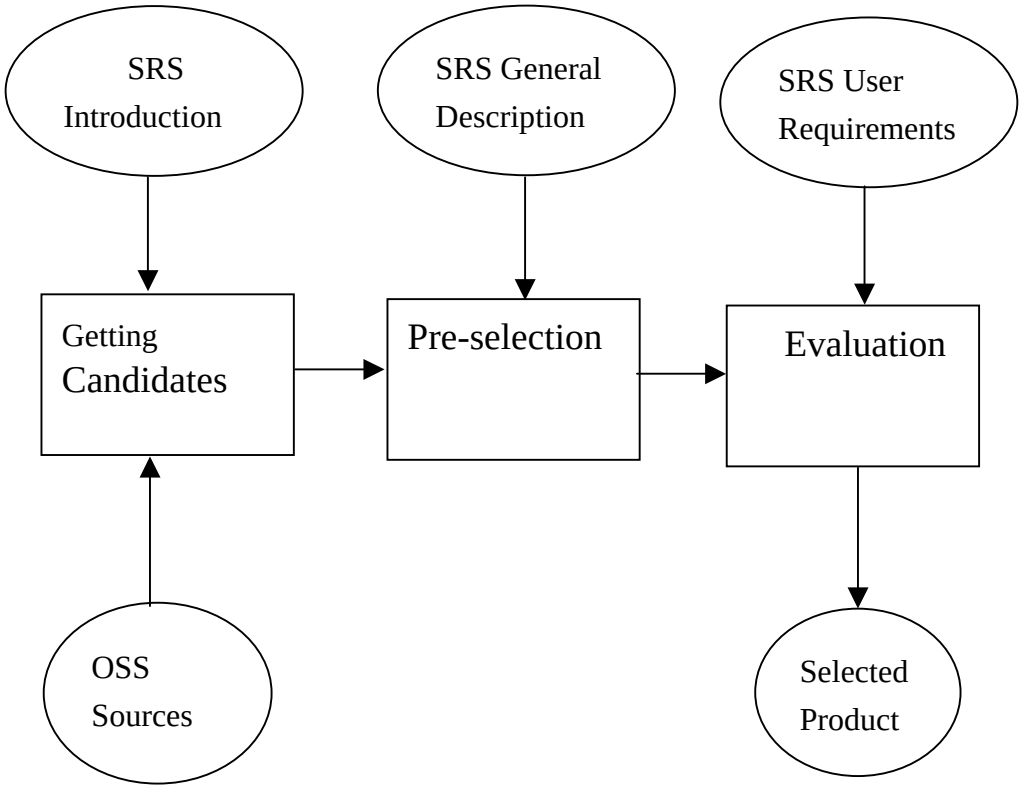


Figure 6. RE for OSS selection model

In Figure 6, the ellipses are inputs and outputs. The input contains OSS sources and SRS introduction, general description and user requirements, which come from the SRS of the product. The output is the selected product. The rectangles are processes of the model. There are three stages, getting candidates, pre-selection and evaluation.

5.1. Getting candidates

When evaluating software, a small list of candidates is needed. The process of getting candidates will complete this object. In Figure 6, it shows that, there are two inputs of this process, OSS sources and SRS introduction. The SRS introduction provides the business requirements of the product. According to the business requirements from SRS Introduction, we select the candidates from a variety of sources. There are five approaches that can be used to find candidates [Golden, 2005]:

- Search Open Source Project Portals – for example SourceForge and Fresh-Meat[1]
- Search the Web – this can also give pages about projects and user opinions
- Ask Open Source Developers
- Post to Mailing Lists
- Ask Vendors

At last, we will get M (M >= 1) candidates. These M candidates are the input of the next process pre-selection.

5.2. Pre-selection

In order to decrease the evaluation efforts, we do the pre-selection according to the general description of the product. The candidates who do not meet the needs of general description of the product will be eliminated. At last, the proper N (N<=M) candidates are selected. In Figure 6, it shows that, this process has two inputs, one is the M candidates, the other one is the general description which comes from the SRS of the product. At last, we will get N candidates. These N candidates are the input of the next process evaluation.

5.3. Evaluation

The last process of the process model is evaluation. In this process, the N candidates will be evaluated according to the requirements of the product. In Figure 6, it shows that, this process has two inputs, N candidates and user requirements. According to the

[1] FreshMeat: http://freshmeat.net/

user requirements, the N candidates will be evaluated. The out of this process is the selected product that is the highest score candidate.

Linear weighted attribute model is a commonly used approach for software selection. In this model a number of attributes are used and each package gets a performance rating for each attribute. Weights are assigned to the attributes, which defines the compensatory nature of this model. The final score of each package is defined by the equation:

$$Qi = \sum_{j=1}^{m} WjAij$$

[Anderson, 1990]

In this study, we will use this model to evaluate the candidates. Qi is the score of candidate i, Wj is the weight assigned to requirement j and Aij is the score of candidate i for criterion j. Thus the final score for a candidate is the sum of the scores of m requirements. Here, the all requirements come from the user requirements of SRS of the product. The highest score one the selected product. In next sections, we will discuss how to evaluate each requirement.

The weight of each requirement depends on customers' needs. Different customers may have different needs. For example, a single customer who will not modify the source code of a product may think the functionality is more important than the issue of license. However, for enterprise customers who will modify the source code of the product and redistribute the modified version, they may think the license issue is most important.

The score of each requirement depends on how each requirement satisfies the customers' need. In next section, we will discuss how to evaluate each requirement.

5.3.1. Community

The community is the driving force behind an Open Source project. It reflects the activity and other areas of the project such as support and documentation. When we evaluate a community, the following should be taken into account:

Number of posts: How a number of topics and replies are posted per period and all time.

Number of Users: How a number of users have registered, how a number of them are online regularly, and how a number of them are active users?

Response: How soon the user questions can be answered, and the quality of the answers.

Friendliness: How friendly the community is towards each other, especially to newcomers.

We also need check whether or not all posts are kept or archived and whether or not the community has the search functionality. Sometimes, the users can find the answers in the old posts.

5.3.2. Cost

As we have mentioned, the cost is not only purchase cost, but also all cost related to the OSS product, such as the support, maintenance and training. When we evaluate it, we should consider about the TCO of the product. Its score depends on the budget of the user.

5.3.3. Development group

The experienced developers are important factor for guaranteeing OSS quality. When we evaluate the development group, we should check the reputation of it. Do they have any other successful OSS products? Besides, we should know the motivation of them developing the product. Do they develop the product just for fun?

5.3.4. Documentation

Documentation contains user documentation and developer documentation. It can help user to install, use and develop the product. We should check whether the product has the sufficient user documentation and developer documentation on the project's site, and the available developer documents if they are clear on how to develop the software and how to join the developer community [Wheeler, 2007]. Additional documentation may include descriptions of the main features, 'How-Tos' and/or tutorials with instructions [Duijnhouwer and Widdows, 2003].

5.3.5. Functionality

A list of functional requirements for the goal of use of the software can be used to check if the needed functionality is available. In this study, the functional requirements come from the SRS. When comparing functionality, those features that are part of the functional requirements should take priority. If there is something missing, there is always the option to build it.

5.3.6. License

When evaluate the license of a product, we should check whether it is an OSI approved license or not. If the product uses a different license, we have to read it carefully, and check whether it is in line with the intended use.

5.3.7. Lifecycle

Lifecycle is a measurement of a product's stability. When we evaluate it, we should check the age and version number of a product. Besides, if the product is very old, we should check whether it is compatible with the new technology.

5.3.8. Market Share

We should check who are the main users of the product and the percentage of it in the total available market. If the product comes from SourceForge, we should check how a number of time it have been downloaded.

5.3.9. Security

OSS security is the measure of assurance or guarantee in the freedom from danger, risk, etc in an OSS system [Wikipedia- security, 2008]. We should check how a number of bugs are there, and how soon the bugs were fixed in a new version. It can show how serious the project is about the security.

5.3.10. Supporting Service

The supporting service includes free support and paid support. The free support for OSS is in most cases handled by the community. The community's support areas are invaluable resources for solving problems [Golden, 2005]. We should check whether the project has a forum or mailing list providing the free support, and how active they are.

Mature products often have paid support options as well if more help or the security of a support contract is required. Usually, the development group can provide the paid support. However, we also should check whether there are any third parties who have given their opinion about the quality of this support. One of the strong signs of maturity of OSS is the availability of third party support: companies that offer commercial support services for OS products [Duijnhouwer and Widdows, 2003].

5.3.11. Usability

Usability is the measure that assesses how easy user interfaces are to use. When we evaluate it, we should check how easy it is for users to accomplish basic tasks at the first time, and how pleasant it is to use the product.

5.4. Summary

In this chapter, we introduced the process model for OSS selection. It has three processes, getting candidates, pre-selection and evaluation. The candidates come from the OSS sources, such as open source project portals. The introduction of SRS is the criteria for getting candidates. The general description of SRS is the criteria for pre-

selection. The user requirements of SRS are criteria for evaluation. The eleven factors, cost, community, development group, documentation, functionality, license, lifecycle, market share, security, supporting service and usability, form the user requirements of the SRS.

6. Case Studies: Media Player

A number of people often watch movie or listen music on computer. The media player is basic software in our life. Except the Windows Media Player, there are a number of other OSS media players. Parts of them are better than Windows Media Player. In this case, the OSS media players will be tested. The customer and user of it are same person, who is an experienced and skilled computer user. He wants an OSS media player which can plays common types of movie files and music files. In addition, it should be free to download and use.

In Chapter 4 and Chapter 5, the RSR template for OSS selection and the selection model have been defined. Now, we apply them to a real OSS products selection. Media Player is chosen as the target product. First, we will use the SRS template to create the SRS for Media Player selection. Then, according to the SRS, we get the proper candidates list and evaluate all of them. Finally, the highest scoring product will be selected. The SRS for Media Player selection can be found in Appendix C.

6.1. Getting Candidates

As we have mentioned in Chapter 5, there are two inputs, OSS sources and SRS introduction of this process. In this case, we get the all candidates come from Website of Wikipedia searching by “Media Player”. The list of media players is Appendix D. In the introduction of SRS, we get the business requirements, the media player can play video and audio, and must be an OSS product and free to download and use. According to it, we get the following candidates from the whole list. They can be found in Table 6-1.

Candidates	Website
Kantaris Media Player	http://www.kantaris.org/
KMPlayer	http://www.kmplayer.com/forums/index.php?
MPC – Homecinema	http://sourceforge.net/projects/mpc-hc/
Mplayer	http://www.mplayerhq.hu/design7/news.html
VLC media player	http://www.videolan.org/vlc/

Table 6-1 Case study candidates

6.2. Pre-selection

The pre-selection is based on the General Description of SRS for Media Player selection. It uses General Description of Media Player to eliminate those candidates that do not conform the basic requirements of the general description. There are three basic requirements in this General Description of SRS:

- License – The Media Player should be under GPL or GPL compatible license, and free to download.
- Support files – The Media Player can play following types of files, mp3, mp4, avi and Mpeg.
- Support OS – The Media Player can be run on Windows Xp, and Ubuntu/Linux.

According the description of each candidate, we can test whether it conforms the basic functional requirements or not. The description of each candidate can be found in Appendix E. Table 6-2 shows the result of the pre-selection.

Candidates	License	Support files	Support OS	Result
Kantaris Media layer	Yes	No	No	No
KMPlayer	Yes	Yes	No	No
MPC – Homecinema	Yes	Yes	No	No
Mplayer	Yes	Yes	Yes	Yes
VLC media player	Yes	Yes	Yes	Yes

Table 6-2 Pre-selection

If a candidate that conforms the requirement of “Support files”, it will be marked as “Yes” under column “Support files”, else as “No”. If a candidate that conforms the requirement of “Support OS ”, it will be marked as “Yes” under column “Support OS”, else as “No”. If a candidate that conforms the requirement of “License”, it will be marked as “Yes” under column “License”, else as “No”. If a candidate conforms all of the three requirements, it will be marked as “Yes” under column “Result”, else as “No”.

According the result, those candidates which do not conform the two basic requirements will be eliminated. At last, we got two conformed candidates, Mplayer and VLC media player. In next chapter, we will evaluate them according the business

requirements, functional requirements and non-functional requirements, which can be found in SRS for Media Player selection.

6.3. Evaluation

The two candidates resulted by the pre-selection will be evaluated: Mplayer and VLC media player.

Mplayer

MPlayer is a free and OS media player distributed under the GNU GPL. It can run on a number of systems, including Linux and other Unix-like systems, Microsoft Windows and Mac OS X. It is available in English, Hungarian, Polish, Russian and Spanish**.** The initial release was in 2000.

VLC media player

VLC media player is an OSS cross-platform media player and streaming server distributed under the GNU GPL. It can run on Mac OS X, Windows, BeOS, Linux, FreeBSD and WinCE. The initial release was in 2001.

Scores ranging from 1 to 10 are given on all criterions for each product. One is given when it does not fulfil any of the wanted characteristics of the criterion, and ten when that the software ideally complies with the criterion. The weights of each criterion are distributed by the importance of them. The evaluating party can define the importance of each criterion according his/her requirements. If the ideal product's score is 1000, then every criterion for the product' score is 10. Therefore, the sum of the weights should be 100.

In this case study, all criterions come from business requirements, functional requirements and non-functional requirements. The most important criterion is functionality. It was give a weight of 30 and the remaining 70 points were distributed evenly by level of importance, resulting in the following weight distribution:

- Cost – 10
- Community– 15
- Documentation – 15
- Functionality – 30
- Lifecycle – 5
- Usability – 25

Cost

If the candidate is free to download, it will be scored 10. Else, the score depends on the cost of it. Both of the two products are free to download.

Community

If the candidate does not have forums or mailing lists, it will be scored 0. Else, the score is according the activity of the community, such as the number of topics and/or posts in forums or mailing lists. The Mplayer has four kind of mailing lists, General, for users, for developers and other. It does not have user Forum. The VLC media player does not have the mailing lists. It has user Wiki and Forum, and the forum is very active.

Documentation

The score of documentation depends on whether the candidate has the Wiki or tutorial. The Mplayer has English supporting material and part of Chinese supporting material. The VLC media player has German, English, Spanish, French and Italian supporting material.

Functionality

In order to check the functionality, the two candidates should be downloaded. Then, evaluate each candidate according to the functional requirements. The Mplayer satisfies part of the functional requirements, and the VCL media player can satisfies all the functional requirements.

Lifecycle

If the candidate has more than two years old, it will be scored 10. Else, it will be scored [10 – (24-months)*0.1]. Both of them are more than two years old and release regularly.

Usability

The score of usability depends on the user's personal habit. The Mplayer is command user interface media player, it is difficult to use for the inexperienced users. The VCL has a good graphic user interface, and it is easy to use and learn. The scores are shown in table 6-3.

Criterion	Weight	Mplayer	VCL media player
Cost	10	10	10
Community	15	8	9
Documentation	15	8	9
Functionality	30	6	10
Lifecycle	5	10	10
Usability	25	7	9
Total scores		845	945

Table 6-3 Scores

In table 6-3, we can find that the VCL media player is more competitive than the Mplayer. Therefore, the final selected product is VCL media player.

6.4. Results

In the previous chapters, a SRS template and a model were defined to use for OSS selection. In this chapter, the case study that was performed using that template and model were described. The goal of the case study was to see whether the template and the model are useable for real software.

The SRS template was used to create SRS for media player selection. Through the model, the requirements of the SRS were used to evaluate media player, starting with 5 candidates, and finally evaluating two candidates. According to the user requirements of SRS, VCL media player is the final selected product. The template was well applicable to these candidates. It has complete and unambiguous requirements. The requirements are consistent. The structure of SRS makes the requirements easily to modify.

7. Conclusions

This thesis studied the OSS from RE point of view in order to create a SRS for OSS selection. It includes a case study to test this SRS template on real software. In this final chapter the results of this research are discussed by answering the research question and subquestions posed in the first chapter.

7.1. Research Results

In this thesis the unique characteristics of OSS were investigated to construct a model for OSS selection. In the first chapter the research question and subquestions were formed. In order to answer the research question, the subquestions should be answered firstly. The answers to those subquestions will now be given by briefly recapitulating the main issues addressed in this thesis.

The first subquestion is: **What factors affect the OSS selection?**

A number of characteristics were found in relation to OSS selection. The following eleven factors were found using literature on OSS:

Cost

The cost of OSS product is not only the purchase cost but also any costs related to the product, such as support and training costs.

Community

The community of an OSS project is the driving force behind the project.

Development Group

Skilled and experienced developer is a guarantee for quality of OSS product.

Documentation

Documentation is very important for user to use or modify the OSS program.

Functionality

User can download the OSS product and evaluate it. The functionalities can be checked during the evaluation.

License

The license needs to fit with the intended use. Usually, the common licenses are preferable, such as GNU GPL and LGPL.

Lifecycle
The lifecycle is measurement of a product's stability. The successful product should release regularly.

Market Share
Market share reflects how popular the product is. For a popular OSS product, there are a number of third parties can provide supports for it.

Security
As the source code is available for both user and attacker. Security needs to be considered seriously.

Support
Support contains free support and paid support. The common free support includes mailing list and user forum. Usually, the development group provides the paid support for its product.

Usability
A highly usability program is easy to learn and use. It is an important factor affecting OSS selection.

The second subquestion is: **How these factors are reflected in SRS template?**

The answer to this question can be found in the user requirements of SRS template. All factors are grouped into three kinds of requirements, business requirements, functional requirements and non-functional requirements. Business requirements include the factors of cost and market share. Functional requirements include the factors of functionality. Non-functional requirements include community, development group, documentation, functionality, license, lifecycle, security, support and usability.

The third subquestion is: **How to evaluate these factors?**

Using the selection model, this question can be answered. For each factor a description of the selection process and the information necessary to establish a score can be found in this thesis. According to the Anderson's model, we can get the total score of each candidate.
Now, the main question can be answered:

How to specify requirements for OSS selection?

The answer to this question is using the SRS template for OSS selection. It was defined using various literature and was well applicable, and followed to come to a satisfactory result in the case of Media Player.

7.2. Contribution

In this thesis, a SRS for OSS selection and selection model were produced. They can help the users, who want to use OSS but are not familiar with the characteristics of OSS, to select proper software from a list of candidates.

The field of OSS in scientific research is still small. With the growing interest of the business world in OSS, the scientific research is growing as well, though it is still behind in many respects. This thesis can add to this research field. It gives insights into OSS from a business use perspective.

7.3. Limitations and Future Work

The study offers a starting point to further the research in the area of OSS selection. This relatively new area does still leave much to explore. There are a number of extended areas that require further consideration and study. In the selection model, each requirement was scored, but it could still use a better way score each requirements. For example, each functional requirement could be given a priority, and a more statistical approach could be taken into account to score the non-functional requirements, such as community. In order to check whether the template works or not, a case study was conducted on Media Player. However, the user of it is single user. As the single user has different requirements from the enterprise user, the template should be tested to see how it works for enterprise user.

There are also other existing models for OSS selection, such as Business Readiness Rating™ (BRR) and Open Source Maturity Model (OSMM). BRR was developed by Spikesource, Carnegie MellonWest and Intel [BRR, 2008]. It is being proposed as a new standard model for rating OSS. It is intended to enable the entire community (enterprise adopters and developers) to rate software in an open and standardized way. OSMM was developed by Navica [2008]. It was developed to help IT procurement managers to better compare and assess OSS. The two models show many similarities to the model that is proposed here. Both of them assess and weight the factors, which affect the OSS selection. Depending upon how well the software meets users needs, each factor is scored. The highest score one is the final selection candidate. However, the OSMM is largely the work of one man, and the BRR favours an evolving community-development model. Whereas the SRS template is designed from the requirements engineering point of view, both single user and multi users can use it.

References

[Anderson, 1990] Anderson. Choice Models for the Evaluation and Selection of Software Packages. *Management Information Systems*, **6** (4), 1990, 123–138.

[Boehm and Papaccio, 1988] Boehm, B. W and Papaccio, P. N, Understanding and Controlling Software Cost. IN: *Proc .of IEEE Transactions on Software Engineering*, Vol. 14, No. 10, 1462-1477.

[Barbara and Bernd 2006] Barbara Paech and Bernd Reuschenbach, Open Source Requirements Engineering. In: *Proc. of the 14th IEEE International Requirements Engineering Conference.*

[BRR. Business Readiness Rating for Open source; A Proposed Open Standard to Facilitate Assesment and Adoption of Open Source Software, RFC1, 2005. http://www.openbrr.org/docs/BRR_whitepaper_2005RFC1.pdf. (checked 23,05,2008)

[Chen, 2006] Shun-ling Chen, *Free And Open Source Software Licensing Primer*. UNDP-APDIP, Elsevier, 2006, p. 29.

[C. Jones, 1996] C. Jones, *Applied Software Measurement: Assuring Productivity and Quality*, McGraw-Hill, New York, 1996.

[Computer Economics, 2008] Computer Economics website, http://www.computereconomics.com/ (Checked 25.05.2008)

[Cooper, 2003] Charles Cooper, *Microsoft shows Linux some respect*. http://zdnet.com.com/2100-1104-981552.html (Checked 15.05.2008)

[COTS-Wikipedia, 2007] Wikipedia website, Commercial off- the-shelf, http://en.wikipedia.org/wiki/Commercial_off-the-shelf (Checked 25.11.2007)

[Cowan, 2003] Crispin Cowan, Software Security for Open-Source Systems. *In: Proc. of Security & Privacy Magazine, IEEE,* **1**(1): 2003, 38–45.

[Davies and Hsia, 1994] A. M. Davies and P. Hsia, Giving Voice To Requirements Engineering, In: *Proc. of IEEE Software, Vol. 11, No. 2*, March 1994, 12-15.

[Duijnhouwer and Widdows, 2003] Duijnhouwer and C. Widdows Capgemini, Open Source Maturity Model. http://pascal.case.unibz.it/retrieve/1097/GB_Expert_Letter_Open_Source_Maturity_Model_1.5.31.pdf (Checked 12.4.2008)

[Donn Le Vie, 2007] Donn Le Vie, Jr, *Writing Software Requirements Specifications*. http://www.techwr-l.com/techwhirl/magazine/writing/softwarerequirementspecs.html (Checked 15.06.2007)

[Firefox, 2008] Mozilla Firefox website. http://www.mozilla.com/en-US/firefox/ (Checked 12.5.2008)

[FFmpeg , 2008] Ffmpeg website. http://ffmpeg.mplayerhq.hu/ (Checked 12.11.2007)

[FOSS, 2008] Wikipedia website, Free Software and Open Source Software. http://en.wikipedia.org/wiki/Free_and_open-source_software (Checked 12.4.2008)

[Free software, 2008] Free software, 2008. http://www.fsf.org/licensing/essays/free-sw.html

[FSF 2007] Free Software Foundation. http://www.fsf.org/ (Checked 20.7.2007)

[GNU 2007a] GNU website, Categories of Free and Non-Free Software. http://www.gnu.org/philosophy/categories.html (Accessed 20.7.2007)

[GNU, 2007b] GNU website, Free Software Definition. http://www.gnu.org/philosophy/free-sw.html (Checked 20.7.2007)

[GNU, 2007c] GNU website, Copyleft. http://www.gnu.org/copyleft/copyleft.html (Checked 15.9.2007)

[GNU, 2007d] GNU website, Why Free Software Is Better Than Open Source. http://www.gnu.org/philosophy/free-software-for-freedom.html (Checked 20.10.2007)

[GNU, 2007e] GNU website, Why Free software is better than Open source software? http://www.gnu.org/philosophy/free-software-for-freedom.html (Checked 20.10.2007)

[Golden, 2005] B. Golden, *Succeeding with Open Source*. Addison-Wesley Pearson Education, 2005. ISBN 0-321-26853-9, 94-96.

[Gonzalez-Barahona, 2000] Jesus M. Gonzalez-Barahona, A Brief History of Open Source Software. http://eu.conecta.it/paper/brief_history_open_source.html (Checked 20.10.2007)

[GPLv3, 2007] GNU General Public License Version 3. http://www.fsf.org/licensing/licenses/gpl.html (Checked 10.11.2007)

[Hart, 2003] Timothy D. Hart, Open Source in Edution. http://portfolio.umaine.edu/~hartt/OS%20in%20Education.pdf (Checked 05.09.2007)

[IBM, 2008] Open Source at IBM. http://www-03.ibm.com/linux/opensource/index.shtml (Checked 20.5.2008)

[IDABC, 2007] IDABC, The A number of Aspects of Open Source. http://ec.europa.eu/idabc/en/document/1744/468 (Checked 20.11.2007)

[IEEE Std 830, 1998] IEEE Std 830, IEEE Recommended Practice for Software Requirements Specifications. In: Proc. of *Software Engineering Standards Committee of the IEEE Computer Society*, p. 31.

[Joch, 2004] Alan Joch, The Real Cost of Open Source. http://www.fcw.com/print/10_43/news/84599-1.html (Checked 20.05.2008)

[Kasse Initiatives, 2004] Kasse Initiatives, LLC, Requirements Engineering. http://www.dtic.mil/ndia/2004cmmi/CMMIT1Mon/Track1IntrotoSystemsEngineering/KISE06RequirementsEngineeringv3.pdf (Checked 21.05.2008)

[Kotonya and Sommerville, 1998] Grald Kotonya and Ian Sommerville, *Requirments Engineering*, John Wiley & Sons, UK, 1998, p.8.

[Kruchten, 2000] P. Kruchten, *The Rational Unified Process.* Second Edition, Addison-Wesley, 2000, p.298.

[Kumar, 2007] Vimal Kumar, Selection and Management of Open Source Software in Libraries. In: *Proc. of 5th International Convention on Automation of Libraries in Education and Research Institutions*, Chandigarh, India, 1-5.

[LC, 2007] David A. Wheeler, The Free-Libre / Open Source Software (FLOSS) License Slide. http://www.dwheeler.com/essays/floss-license-slide.html (Checked 28.10.2007).

[LGPLv3, 2007] GNU Lesser General Public License version 3. http://www.fsf.org/licensing/licenses/lgpl.html (Checked 10.11.2007)

[Linux, 2008] Linux Operation System Website. http://www.linux.org/ (Checked 10.11.2007)

[Macaulay, 1996] Macaulay, L. A, *Requirements Engineering*, Springer-Verlag.

[Massey, 2002] Bart Massey, Where Do Open Source Requirements Come From (And What Should We Do About It)? In *Proc. 2nd Workshop On Open-Source Software Engineering*, Orlando, FL, May 2002

[Metcalfe, 2008] Randy Metcalfe, Top Tips for Selecting Open Source Software. http://www.oss-watch.ac.uk/resources/tips.xml (Checked 13.04.2008)

[Morgan, 2004] Eric Lease Morgan, Open Source Software in Liraries. http://infomotions.com/musings/biblioacid/ (Checked 22,10.2007)

[Morgan, 2004] Eric Lease Morgan, Open Source Software for Libraries in 30 minutes. http://infomotions.com/musings/oss-in-thirty-minutes/ (Checked 22.10.2007)

[Navica, 2008] Navica Open Source Maturity Model. http://www.navicasoft.com/pages/osmm.htm (Checked 13.04.2008)

[NETC, 2007] Northwest Educational Technology Consortium, This Brief History Illustrates the Origins and Major Events in Open Source. http://www.netc.org/openoptions/background/history.html (Checked 26.7.2007)

[NETC, 2008] Northwest Educational Technology Consortium, 2008. Total Cost of Ownership. http://www.netc.org/openoptions/pros_cons/tco.html (Checked 12.6.2008)

[Nichols and Twidale] David M. Nichols and Michael B Twidale, The Usability of Open Source Software. http://www.firstmonday.org/Issues/issue8_1/nichols/ (Checked 20.5.2008)

[Nokia, 2008] Open Source at Nokia. http://opensource.nokia.com/ (Checked 20.5.2008)

[Non-functional requirements, 2008] Non-functional Requirements, Wikipeida Website. http://en.wikipedia.org/wiki/Non-functional_requirement (Checked 12.5.2008)

[Nuseibeh and Easterbrook 2004] Nuseibeh and Easterbrook, RE: A Roadmap. In: P*roc. Proceedings of the Conference on The Future of Software Engineering*, 35 – 46

[OSS-Definition, 2007] OSI Website, The Open Source. Definition. http://www.opensource.org/docs/definition.php (Checked 20.7.2007)

[OSI, 2008] Open Source Initiative. http://www.opensource.org/ (Checked 12.04.2008)

[Orzech, 2002] Orzech, D, CIN: Linux TCO: Less than Half the Cost of Windows. http://linuxtoday.com/it_management/2002100801926NWBZMR (Checked 13.05.2008)

[O'Riordan, 2006] Ciaran O'Riordan, How GPLv3 Tackles License Proliferation. http://www.linuxdevices.com/articles/AT7188273245.html (Checked 23.11.2007)

[Pavliced 2000] Pavlicek Russell, Embracing Insanity: Open Source Software Development. Indianapolis: SAMA, 2000.

[Perens, 2008] Bruce Perens. http://en.wikipedia.org/wiki/Bruce_Perens (Checked 23.11.2007)

[Raymond, 1998] Eric S. Raymond (1998-02-08). Goodbye, “free software”; hello “open source”.

[Richard 2007] Richard Stallman, Why “Open Source” misses the point of Free Software. http://www.gnu.org/philosophy/open-source-misses-the-point.html (Checked 30.10.2007)

[Richard, 2007] Stallman Richard 2007. Why “Free software is better than Open Source”. *Philosophy of the GNU Project*.

[Robinson and Pawlowski, 1999] W.N. Robinson and S.D. Pawlowski, Managing Requirements Inconsistency with Development Goal Monitors. In: *Prco. of IEEE Trans. Software Eng.* 25, 1999, 816-835.

[SC, 2007] Software Categories. Categories of Free and Non-Free Software. http://www.gnu.org/philosophy/categories.html (Checked 20.7.2007)

[Scacchi, 2002] Walt. Scacchi, Understanding the Requirements for Developing Open Source Software Systems. In: Prco. *Of Software*, volume 149, pages 24–29, 2002.

[Software Bug, 2008] Software bug, Wikipedia website. http://en.wikipedia.org/wiki/Software_bug (Checked 25.2.2008).

[Software Requirements Inc, 2008] Software Requirements Inc website, What Makes a Good Software Requirements Specification. http://www.softreq.com/answer.cfm?question_id=4 (Checked 12.06.2008)

[Sommerville and Sawyer, 1997] Ian Sommerville and Pete Sawyer, *Requirements Engineering*: A Good Practice Guide, Wiley, 1997.

[Stallman, 2002] Richard M. Stallman, Why “Free Software” is better than “Open Source”. http://www.gnu.org/philosophy/free-software-for-freedom.html#relationship (Checked 25.11.2007).

[Stallman, 2007] Richard M. Stallman, Stallman explains licence compatibility while discussing GPLv3.

[TCO, 2008] Northwest Educational Technology Consortium, Total Cost of Ownership. http://www.netc.org/openoptions/pros_cons/tco.html (Checked 20.4.2008)

[Tim O'Reilly, 1999] Tim O'Reilly, *Open Source: Voices from the Open Source Revolution*, 1st edition January 1999.

[Weber, 2004] S. Weber, *The Success of Open Source.* Harvard University Press, 2004. ISBN 0674012925.

[Wheeler, 2007] David A. Wheeler, How to evaluate Open Source Software / Free Software Programs. http://www.dwheeler.com/oss_fs_eval.html (Checked 28.10.2007).

[Wheeler, 2007] David A. Wheeler, Open Source Software / Free Software (OSS/FS) References. http://www.dwheeler.com/oss_fs_refs.html (Checked 28.10.2007).

[Wikipedia- security, 2008] Wikipedia Website, Open Source Software Security. http://en.wikipedia.org/wiki/Open_source_software_security (Checked 20.3.2008)

[Wikipedia - Software, 2007] Wikipedia Website, Computer software. http://en.wikipedia.org/wiki/Software (Checked 24.9.2007)

[XFree86 2007] XFree86, The XFree86 Project, Inc. http://www.xfree86.org/ (Checked 1.11.2007)

[Yeates, 2005] Stuart Yeates, Open Source Software and Security http://www.oss-watch.ac.uk/resources/security.xml (Checked 10.01.2008).

[Zave, 1997] P. Zave, Classification of Research Efforts in Requirements Engineering, *ACM Computing Surveys*, **29**(4), 1997.

[Zhang, 2007a] Zheying Zhang, "Effective Requirements Development - A Comparison of Requirements Elicitation techniques", *INSPIRE*, 2007.

[Zhang, 2007b] Zheying Zhang, Requirements Engineering Lecture Notes, University of Tampere, Fall 2007, http://www.cs.uta.fi/re/L2_fall.pdf (Checked 24.3.2008)

Appendix A: IEEE Requirements document structure [IEEE Std 830, 1998]

1. Introduction
    - 1.1 Purpose
    - 1.2 Document Conventions
    - 1.3 Intended Audience and Reading Suggestions
    - 1.4 Product Scope
    - 1.5 References
2. Overall description
    - 2.1 Product perspective
    - 2.2 Product functions
    - 2.3 User classes and characteristics
    - 2.4 Operating environments
    - 2.5 Design and implementation constraints
    - 2.5 Assumptions and dependencies
3. External interface requirements
    - 3.1 User interfaces
    - 3.2 Hardware interfaces
    - 3.3 Software interfaces
    - 3.4 Communications interfaces
4. System Features
    - 4.x System feature X
        - 4.x.1 Description and priority
        - 4.x.2 Stimulus/Response sequences
        - 4.x.3 Functional requirements
5. Other non-functional requirements
    - 5.1 Performance requirements
    - 5.2 Safety requirements
    - 5.3 Security requirements
    - 5.4 Software quality attributes
    - 5.5 Business rules
    - 5.6 User documentation
6. Other requirements

Appendix A: Glossary

Appendix B: Analysis Models

Appendix C: To-Be-Determined List

Appendix B: SRS template for OSS selection

Software Requirements Specification
For
<Product>
Selection

Version: X.X
Author: XXX
Date:(mm/dd/yyyy)

Contents

Revision History

Name	Date	Reason For Changes	Version

1. Introduction

The following subsections of the Software Requirements Specifications (SRS) should provide an overview of the entire SRS.

1.1. Purpose

Specify the purpose of this SRS and its intended audience.

1.2. Scope

1. *Explain what the wanted product will, and if necessary, will not do. This should be an executive-level summary. Do not enumerate the whole requirements list here.*
2. *Identify the all candidate products to be selected by name*

1.3. Definitions, acronyms and abbreviations

Provide the definitions of all terms, acronyms, and abbreviations required to properly interpret the SRS. This information may be provided by reference to one or more appendixes in the SRS or by reference to other documents

1.4. References

List any other documents or Web addresses to which this SRS refers. Provide enough information so that the reader could access a copy of each reference, including title, author, version number, date, and source or location.

2. General Description

Describe the main features of the selected product should have.

2.1. Product Features

Summarize the major features the product contains or the significant functions that it performs or lets the user perform. Details will be provided in Chapter 3, so only a high level summary is needed here.

2.2. Operating Environment

Describe the environment in which the software will operate, including the hardware platform, operating system and versions, and any other software components or applications with which it must peacefully coexist.

2.3. User Characteristics

Describe those general characteristics of the intended users of the product including educational level, experience, and technical expertise. Their experience and comfort with technology will actually influence selection of the product.

3. User Requirements

Cover business, functional and non-functional requirements.

3.1. Functional Requirements

Provide a summary of the functionalities that the selected software can perform. The functionalities are described in the language of the customer.

For clarity:

1. *The functions should be organized in a way that makes the list of functions understandable to the customer or to anyone else reading the document for the first time.*
2. *Textual or graphic methods can be used to show the different functions and their relationships.*

3.2. Non-functional Requirements

There are a number of attributes of software that can serve as non-functional requirements. The following items provide a partial list of examples.

Community,
Developers group,
Documentation,
License,
Lifecycle,
Security,
Supporting service,
Usability,

4. Appendix

The appendixes are not always considered part of the actual SRS and are not always necessary. They may include

a) Sample input/output formats, descriptions of cost analysis studies, or results of user surveys;

b) Supporting or background information that can help the readers of the SRS;

c) A description of the problems to be solved by the software;

d) Special packaging instructions for the code and the media to meet security, export, initial loading, or other requirements.

When appendixes are included, the SRS should explicitly state whether or not the appendixes are to be considered part of the requirements.

Appendix C: SRS for Media Player Selection

Software Requirements Specification
For
<Media Player>
Selection

Version: 0.1
Author: Ying Yang
Date:(06/06/2008)

Contents

Revision History

Name	Date	Reason For Changes	Version

1. Introduction

The following subsections provide an overview of the entire Software Requirements Specification (SRS) for Media Player selection.

1.1. Purpose

The purpose of this SRS is specifying the requirements of a customer wanted an OSS Media Player. It is free to download and use. According to those requirements, all Media Player candidates will be evaluated. And the highest score one is the final selected product.

The intended audience includes single customer and user.

1.2. Scope

This is a requirements specification for selecting OSS product. It describes the requirements, which need to be evaluated during the selection.

The all candidate products are:

Kantaris Media Player,
KMPlayer,
MPC – Homecinema,
Mplayer,
VLC Media Player.

1.3. Definitions, acronyms and abbreviations

GUI: Graphic user interface

OS: Operation System

Single Customer: The customer who will not buy the commercial supports.

SRS: Software Requirements Specification

1.4. References

Kantaris Media Player: http://www.kantaris.org/
KMPlayer: http://www.kmplayer.com/forums/index.php?
MPC – Homecinema: http://sourceforge.net/projects/mpc-hc/
Mplayer: http://www.mplayerhq.hu/design7/news.html
VLC media player: http://www.videolan.org/vlc/

2. General Description

Describe the main features of the selected product should have.

2.1. Product Features

The Media Player can play following types of file, mp3, mp4, avi and Mpeg.

2.2. Operating Environment

The Media Player can run on Windows XP and Ubuntu/Linux OS

2.3. User Characteristics

The user has experience with computer, and has a computer at home.

3. User Requirements

Cover functional and non-functional requirements.

3.1. Functional Requirements

3.1.1. Play a file

The Media Player can play following types of file, mp3, mp4, avi and Mpeg.

3. 1. 2. Play a CD/DVD/VCD

The Media Player can play CD/DVD/VCD from a drive.

3.1.3. Play a network stream

The Media Player can play WebRadio and WebTV.

3.1.4. Play from an acquisition card

The Media Player can play include webcams card.

3.1.5. Playlist

- The Media Player can store a list of several files to play one after the other
- The Media Player allows user to append an item at the end of the playlist (its playback won't start immediately), to save the playlist as a M3U or PLS file, or to import a playlist file.
- The Media Player allows you to sort the playlist according to several criteria.

3.1.6. Hotkeys

The Media Player supports the hotkeys:

Jump 10 seconds backwards: Alt + Left

Jump 10 seconds forwards: Alt + Right

Jump 1 minute backwards: Ctrl + Left

Jump 1 minute forwards: Ctrl + Right

Quit: Alt + q or Alt + F4

Volume up: Ctrl + Up

Volume down: Ctrl + Down

3.2. Non-functional Requirements

3.2.1. Community

The Media Player should have active Mailing list and Forum.

.3.2.2. Developers group

None.

.3.2.3. Documentation

The Media Player should Wiki and tutorial.

.3.2.4. License

The Media Player should be under GPL license, or GPL compatible license.

3.2.5. Lifecycle

The Media Player should have more than 2 years old.

3.2.6. Security

None

3.2.7. Support

No special support.

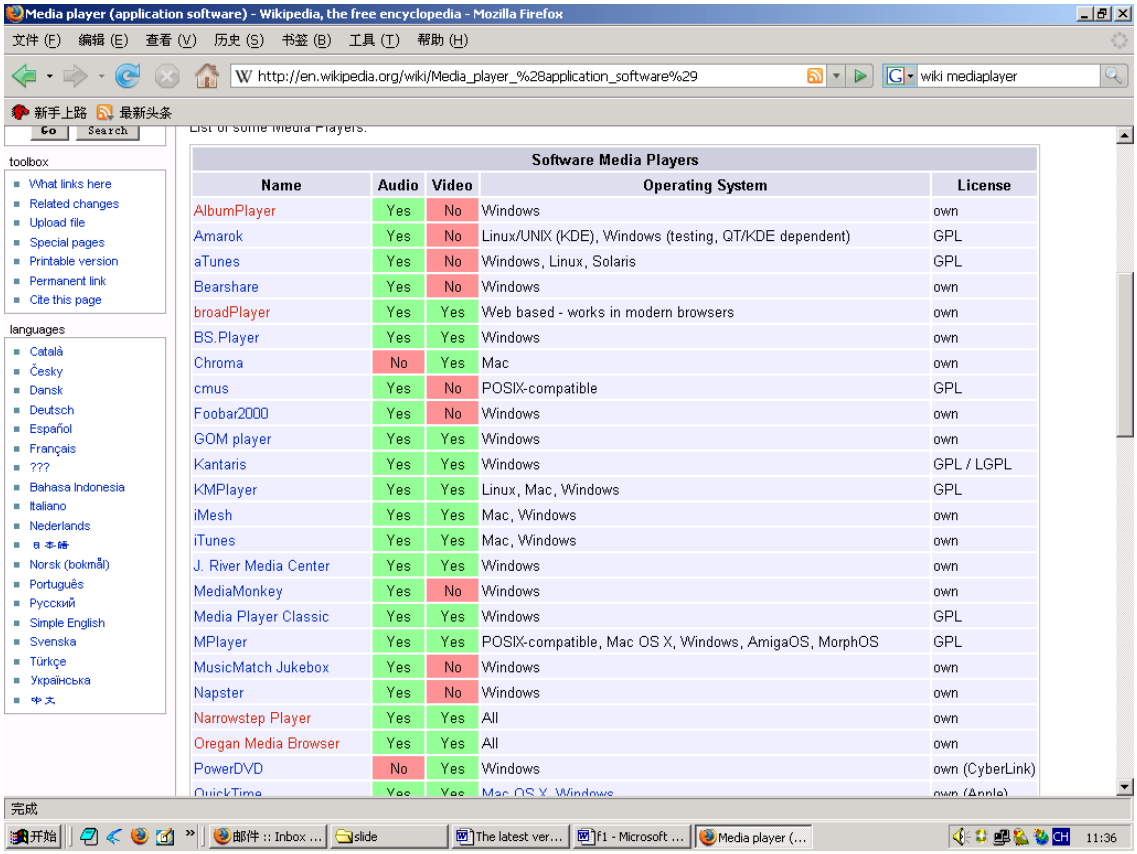
3.2.8. Usability

The Media Player should have friendly graphic user interface (GUI) not command line interface. The user can control it by mouse.

4. Appendix

None.

Appendix D: List of Media Players



Appendix E: The description of each candidate

Kantaris Media Player

Support files: AVI, MPEG, MGEG-AVC, WMV, MOV, MKV, quicktime, matroska, divx, xvid, H264, MP3, WMA and OGG.

Support OS: Windows 2000, XP and Vista.

KMPlayer

Support files: AVI, ASF, WMV, AVS, FLV, MKV, MOV, 3GP, MP4, MPG, MPEG, DAT, OGM, VOB, RM, RMVB, TS, TP, IFO, NSV, MP3, AAC, WAV, WMA, CDA, FLAC, M4A, MID, MKA, MP2, MPA, MPC, APE, OFR, OGG, RA, WV, TTA, AC3 AND DTS.

Support OS: Windows 2000/ XP/Vista

MPC – Homecinema

Support files: AVI, OGM, MK, MPG, VOB, MP4, 3GP, MP3, OGG, MKA, MP4 and AAC

Support OS: Windows 95,98,NT, XP and Vista.

MPlayer

Support files: MPEG, AVI, ASF/WMV, QuickTime/MOV, VIVO, FLI, RealMedia, NuppelVideo, MP4, yuv4mpeg, FILM, RoQ, OGG/OGM, SDP, PVA, NSV, Matroska, NUT, GIF, MP3, OGG/OGM (Vorbis), CD audio and XMMS.

Support OS: Microsoft Windows, Mac OS X, Linux and other Unix-like systems.

VLC media player

Support files: MPEG (ES, PS, TS, PVA, MP3), AVI, MP4 / MOV / 3GP, FLV (Flash), ASF / WMV / WMA and WAV (including DTS).

Support OS: Windows 2000, Xp and Vista, Mac OS, Debian/Linux, Ubuntu/Linux, Rethat/Linux and BeOS.

The all information comes from the homepage of each candidate.