Project Report

Vilma Rudžionytė, Viktoras Čiumanovas,
Darius Damalakas, Martynas Kriaučiūnas

May 27, 2005

Figure 1: The Spirit Horse

Contents

Chapter 1

Introduction

Figure 1.1: The Sky moves

1.1 Abstract

The study project report answers the question “how to make commercial free software?”. See Project Audience for short-readers.

1.2 Project Audience

Nevertheless the project is made as a study activity at RHS (and thus will be read by examinators and censors), the main target audience is for people who wish to get a good overview of a Free Software and Business world areas and how to operate in these two.

For people who read this project report on their personal interest, we suggest reading the project outline first on page 6.

1.3 Project Philosophy

The project philosophy drives our motives and logic used in throughout the project. Thus, we say that:

We (the project team) have a strong determination that it is as necessary to know one area in detail, as it is important to know the same area's global view.

We think that to know the rules is not less important than to know how to create the rules.

We think that rules have a property of "naturalness". This "naturalness" could be understood as a kind of "rationality". If the people would be perfect, they would not need to write down the rules, they would simply know them by heart. Or deduce them using simple logical operations based on facts and based on feelings. As a contrary, trees do not have rules. It is we, homo sapiens, who observe their behavior and say that some their properties are rules. But giving a name to some phenomenon does not affect the phenomenon (at least directly). People gave name to what we call nowadays a hurricane, but it did not change because of that we had called it a hurricane. It is not the hurricane which has changed, it is the people who change theirselves upon acting. Thus, we observe how we are affected by our own rules and the rules already existent in our environment.

Nothing exists for just simply "Because". Everything exists for the *Purpose*, and the purpose is the result of the *Wish*. Our wish is to seek and learn. This way we receive a perpetual joy in our lifes. Thus we applied to study at educational institutions. And thus we want to thank our Mothers and Fathers for we have our gift of dreaming, a gift of producing wishes. We are gratefull

for a chance of coming to Denmark and coming to Roskilde Handels Skole. We were accepted as rightfully students, eventhough we come from a small beloved country Lithuania. For that we thank very much.

Rules do exist for purpose. The purpose of rules are not to adhere them. Rules are rather a property. Without adhering to such properties, no human made system would exist. Unfortunately, the purpose of the rule can not be easily extracted in most cases. We will adhere to any rule where its underlying purpose helps achieve our wishes. Were it not, we will search for help from other people or from ourselves (that is, we will create rules).

Further, we think that to learn we must not investigate the cause after a result has happened. We rather will go two steps behind and understand a cause of a cause. In this way, the result (in cases it is not welcome) can be avoided. In cases some result is welcome, the better understanding of the results cause cause let us know what we are doing.

Adhering to the before mentioned belief, we take as a domain of problem not only the problem question, but the very project report itself is also considered an area of study. This leads us to proper investigation of this report's cause and based on that cause we judge what will be the project report's goal.

1.4 Project Goal

The goal of the project is not only to build up a report and answer a problem question, but also to build a valuable project report (i.e. this document). The valuableness primarily comes from how well it speaks and thus what the readers will learn from it. Surely the readers will differ both in experience and also knowledge level, so writing a good project is quite a challenge to find a balance between getting a good mark (since the project is made as a study activity in Roskilde Handels Skole – RHS), learning something for ourselves, and producing a material so others could learn from it.

Aside from producing a valuable project report, we have a vision, which will be in accordance with our problem definition analysed and solved during the project. Next, we define our system and our work vision.

1.4.1 Vision

Our vision is to ***create a business operating on commercial free software***.

The goal is to create a business model, which would focus on commercialising our software. The goal is to propose possible ways to come specifically into free software business market.

1.4.2 System Vision

Our software vision is to develop a system, which would allow to integrate both the code and system documentation. The goal is that our developed system must be able to extract documentation from the source code and put it into an *on-line* wiki system[1]. The wiki system will be chosen from the already existing ones, and modified to support special needs for documenting system design, such as building class diagrams from the source code.

1.5 What is a Wiki System

Wiki is a web application that allows users to add content, as on an Internet forum, but also allows anyone to edit the content.

Wiki software is a type of collaborative software that runs a Wiki system. It is usually implemented as a server-side script that runs on one or more web servers, with the content generally stored in a relational database management system, although some implementations use the server's file system instead.

The primary difference between wikis and more complex types of content management systems is that wiki software tends to focus on the content, at the expense of the more powerful control over layout

1.6 Project Problem Domain

1.6.1 Project Problem Statement

The projects problem domain can be strictly delimited by formulating it as a question. Thus, everything which falls in helping answering the question will be

[1] Wiki system will be defined in the following section

of our interest.

The problem statement: ***How to make commercial Free Software?***

This is a very abstract question, and can be understood in many ways. The boundary of the question will be limited by our system's vision, which we have developed in parralel with the question. Shortly, our system will be a combination of a Wiki[2] and code documentation generation system. We welcome you to visit our system vision definition sectionon page 4. What is a Wiki system in general is described .on page 4

Next, we de-roll the question into many subproblems:

- What is a free and/or open source software? Is open source the same as free software? Is free software te one, which is given away for zero price? Is free software suitable for making profits? Is open-source software suitable for making profits? What is a software license?
- What is a business models? What are business models' constitutant parts?
- Are there any open source/free software business models? What are traditional business models?

There's quite a lot of questions. However, one can easily notice they easily and rather nicely fall under three main categories: ***Free software***, ***Business Models*** and ***wiki system design***.

Project problem domain is partly influenced by our study curriculum.

1.6.2 Study Curriculum

The project coverage is also formed by the courses attended in Roskilde Handels Skole, fourth semester. These are: ***IT strategy and Business Models*** and ***Advanced Programming***.

Here is the outline for the IT course:

- General System Theory, IT – Strategy, IT and research
- Organizational Structure, Organizational Culture

[2] A wiki system is a type of an on-line content management system, where the content can by edited by any visitor

- Porters Framework, The resource-based view
- Strategic Thinking, The business model concept
- Offering Systems: products and services, Value chain, Resource and competence systems
- Fundamentals of Managing Finance, Logistics management

The following outline is of the Advanced Programming part (Java).

- Collections, Recursion, Beans, JUnit and Ant
- Algorithm Correctness, assert + pre/post-conditions
- Language description, Types: casting/boxing, Generic types
- Inheritance, CORBA, Design Patterns, Antipatterns
- Swing, JNI, OODB – Jasmin, J2EE introduction, Smalltalk

1.7 Project Outline

Here we present briefly our way of problem solving. The ultimate goal is to answer the problem question. It is a vast topic, and thus we here subdivide it into parts and present as diagrams. The goal of these diagrams is to explain what possible sub-problems poses every issue in our problem, and what was the course of our investigations; or in other terms – the table of contents summarised in diagrams.

Chapter 3 Software Licensing.

The diagram 1.4 shows that our endeavor – to understand what is the difference between Free Software and Proprietary licenses. In chapter 3 Software Licensing main focus is put on licenses – what are different free software licenses, what is Free Software and Open Source movements.

Chapter 4 Open Source Development. The diagram 1.3 further shows the course of our investigations - chapter 4 Open Source development. This time, the focus is on advantages and disadvantages of making the project in open-source development style. Agile methodologies are closely related to this area, so they are discussed together.

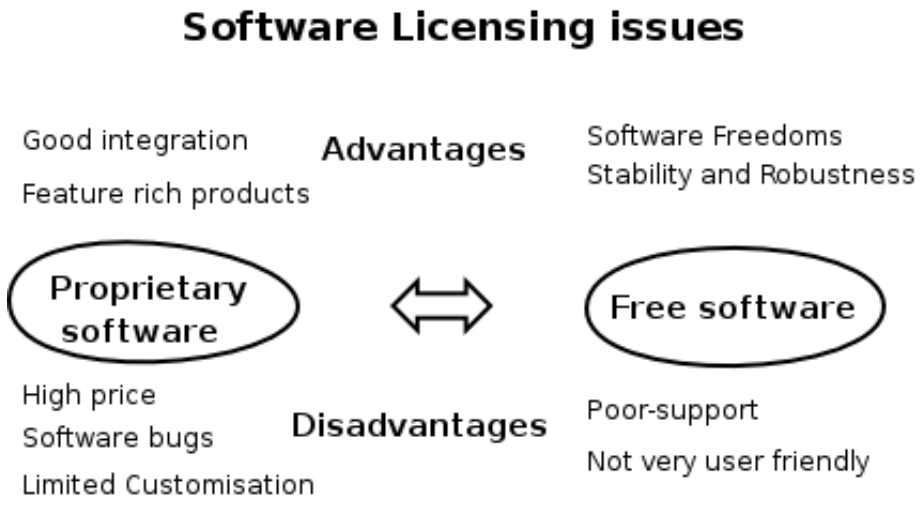


Figure 1.2: Licensing

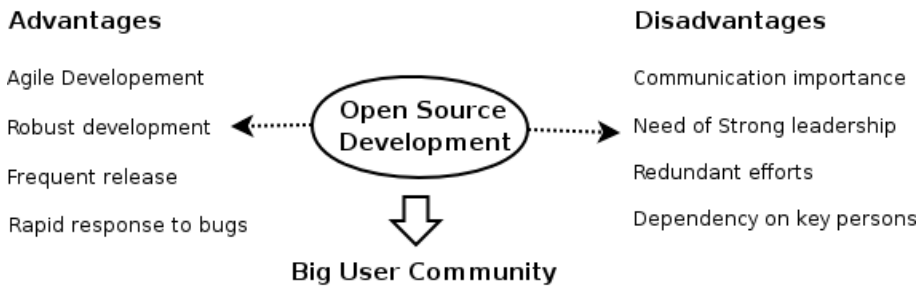


Figure 1.3: Open Source Development

It is rather a misconception to say that strong leadership or dependency on key persons are disadvantages. It would be rather equally not right to say that robust development and rapid response to bugs are inherent qualities of Open Source Development. The diagram merely tries to group them, so it would be easier to grasp, and the real implications will be analysed in detail in the chapter.

Chapter 5 Business Models.

The chapter 5 Business Models goes on to analyse the business area of the project. We explained the chapter with two diagrams.

The first diagram 1.4 summarizes, that if the demand is greater than current competitor product offering, our product has a niche to to spread into. It also

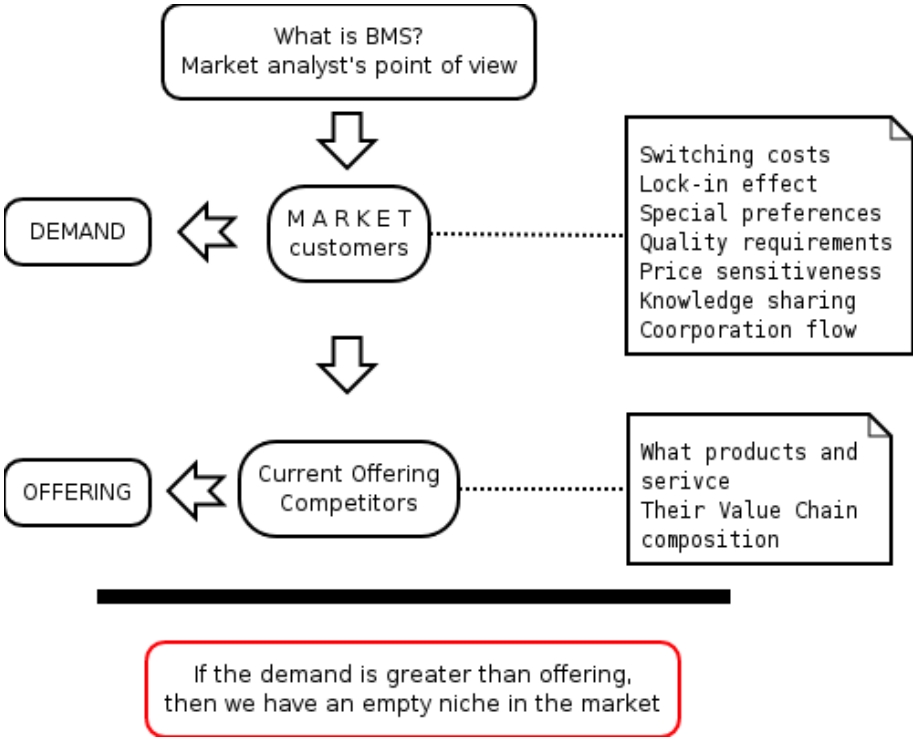


Figure 1.4: Business Analysis

shows what areas influence market and competitor offerings. All maters must be more or less taken into account when giving final solution.

The second figure 1.5 summarizes our project flow. It also emphasizes that choosing a business model is dependent on our product licensing, on whether we choose to do Open Source development and on situation in the market.

Chapter 7 The Sequel.

The last figure 1.6 emphasizes the possible future ways in our project. This chapter says that to make a commercial free software is not enough only to build (or start-building) a system, and choose a business model. Managing resources, activities and structure of organisation, and quality assesment are important issues to consider. However, the project merely glances some of the areas raising only more questions, than we actually answer. This is because the time frame of the project limit to five weeks.

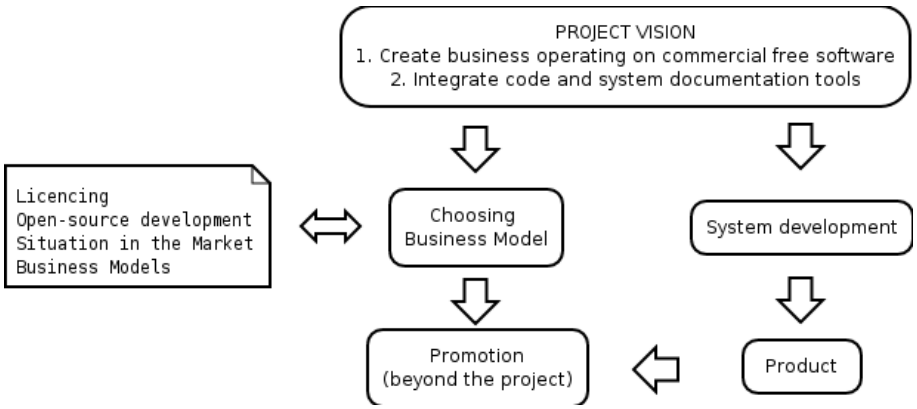


Figure 1.5: Business Model

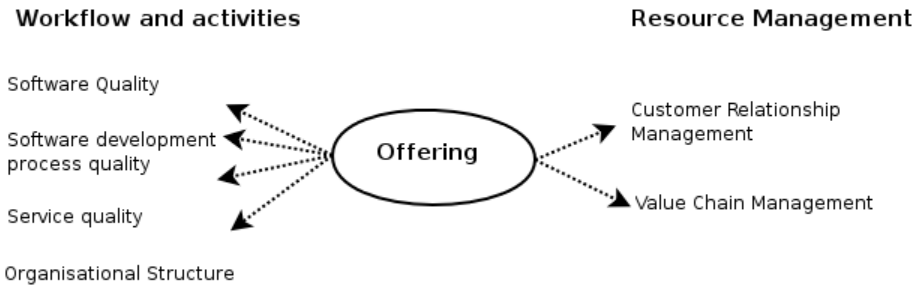


Figure 1.6: Next steps

Chapter 2

ProjectStart

Figure 2.1: The Free Spirit

2.1 "Explosion Like a Baloon" Team

The team doing this project is name "Explosion Like a Ballon".

2.1.1 The Name

We chose our team name according to our goals and philosophy.

Firstly, the name captures the very beautiful moment of an explosion, which is at the same time similar to a baloon. If one had seen a slowed-down splash of a water-drop, one knows how many little sparkles fly around in all directions. It is a very truely astounding scenery. If one would observer a gentle rainfall, one would feel and see the hundreds of such little water-drop explosions.

Secondly, explosion means a burst of power, a some kind of energy. This burst is so high and intensive, that it penetrates the close-surroundings in a millionth part of a second massively distorting and shaking all particles.

And lastly, a balloon explodes when it is full of air, full of ordinary day-to-day air, which eventually spending some time inside a rubber cage (balloon), gets saturated with that filthy gum-air. And when it becomes too big, it explodes, releasing nothing but a filthy air and disappearing as though nobody was there.

2.1.2 Members

We are four members, doing the project and writing the project report.

- Vilma Rudžionytė.
- Viktoras Čiumanovas.
- Martynas Kriaučiūnas.
- Darius Damalakas.

2.1.3 Team Roles

The team roles where not set explicitly. Those were accepted naturally as part of ongoing team self-organising. Since everyone has been working on almost all the project, it is possible only to name persons responsible for managing separate parts:

- Programming and system design This part is managed by Martynas. Darius contributed in writing about web application architectures.
- Business Viktoras and Vilma managed together this big part.

- Free Software and Software licensing Darius is working as of his personal interest on this part.

Our team does not have a leader. However, we have a "dogsbody" person (or hard-worker), who is doing all the hard work in building the project report document (probably "the editor"). Darius willingly accepted this role (and later regretted alot).

2.2 Extreme Project Management

Methodologies are not used only because they simply exist. Some exist for purpose to give an advantage of predictability. Others to give an advantage of coordination, speed, or simplicity. The goals vary greatly. We were searching for a methodology which would help us in reaching our goal – i.e. to learn by producing a quality study project report.

Half a year ago, the team (was slightly different, but the core is the same) had studied various formal software development methodologies, including UP, RUP, XP, Merise and many others. A software methodologies analysis study report was made analysing differences between XP and UP. Combination of these two methodologies to take advantage of UP and advantages of XP agility were conveyed. To our surprise, during this project we found a work named "Agile Modeling and the Rational Unified Process (RUP)", which raises the same problem.

During the exam, the work on adding a business analysis steps to RUP was presented. During this project we found an Enterprise UP, which is the same (although much better and broader and professional) as we presented in exam.

In this semester, on behalf of our problem question we had to analyse the other side of formal methodology area – the agile methods. This in addition fills up the gap in our methodology studies.

2.2.1 XBreed

Our need for methodology relates to two areas: one is how to effectively organise and manage our team and our main activity – project report writing; The other

stems from our problem question, i.e. how to manage free (which we will later see is also open-source) software project.

We have studied Agile methodologies as part of this project [1].

We now say, that for the longer-term project (for example if we would really make our system commercial and set up a company), we would take XBreed as a methodology.

There are many points for it: it is an agile method, which suits us well since the team is small and needs agility to be able to adapt to market and inner changes fast. Further, to make a commercial free software, one must make it also open source and possibly develop it in open-source manner; that means our goal is to have a big user/customer community around the software. This community requires agile development.

To sum up, XBreed is actually a well integrated solution between a Scrum – an agile team management framework, and XP – the eXtreme Programming methodology. We will describe both in the Agile Processes section.

2.2.2 Source Control Tool

In our situation, the project management area mainly focuses to coordinating the writing process of the study report. The goal of our study at RHS is to learn, and thus by writing a good project report we will succeed it.

The biggest problem for four member team is to have a one file for project report. This would cause a lot of trouble.

To split a project into separate files brings a problem of how to assemble the project report into one document. However, this problem occurs only when using software office packets, such as Open Office, Star Office, or Microsoft Word. There are other solutions, which we have chosen and are very happy about them and actually solve these problems. These are

- LaTeX – a high-quality typesetting system
- Subversion (also known as svn) – a version control system

Combining these two systems all the coordination problems on writing a single project report document simply just cease to exist. Using LaTeX, we

[1] See 4.5 section on Agile Processes on page 38

can build a high-quality document with the ability to split the documents in as many smaller parts as we want to. Subversion system allows us to work on even the same part of the project and be sure that no severe (i.e. unrecoverable in 10 minutes) conflicts will occur.

Unfortunately, we did not succeed in proper project report writing management.

2.2.3 Iteration Management

We do not have explicit iterations, but the project was driven in a controlled manner, through a tight coordination of team members and under the firm hand of our dogsbody person.

As with most students happen, the significant amount of work is done during the night before the hand-in date. At the time-being (the time this section is written), the due date is two weeks forward. The project at this pace will receive a final form 5 days to deadline, and the last week will be devoted to filling in the gaps, proof reading and stressing the necessary parts for they would appear more vivid that the others.

We did not do a project activities time chart. This is because we are focusing on producing study project report, not a valuable product.

We have two main iterations – the document formation start and document formation end. These milestones gave us guidelines on what are we going to focus. For the first one we were focusing on producing as much relevant material. In the later one, we were building a cohesive document.

In general, our working method is somewhat similar to the one used in Scrum (See section [2]. Every individual team member has his own goals, which he coordinates with others. And until the milestone they drove their works more to the areas they wanted to, and were not influenced alot. It is not the same as Scrum (how could it be with such short timebox?), but we believe in the long run the team would self-organize in a similar way.

[2] 5 about Scrum on page 43

2.2.4 What went Wrong and Could Go Better

We did choose a very efficient and valuable tool, which allowed all team members to create the same document at the same time. However, we did not succeed in an acceptable project management rating.

What Went Wrong

We faced a couple of problems:

1 Not all team members worked at the project all the time.

2 The contributed work to the project report is very different between members.

3 And most important, the biggest effort to the report was put during the last week.

The following figure gives an example of our effor distribution upon the project report. It gives an ideal variant, which we will seek for the next project, and the current situation in our project.

The problems appeared as a result of our different background, since every individual in our team has not only different educational background, but also different view to the world. Yes, the problems any way would arise, however, our mistake is that we ignored this area and did not anticipate it.

It is not quite possible to measure the effort, since different people work in completely different ways. Albeit knowledge between members can be transferred, skills is something which can not be simply learned in a one-month project. We believe that it is possible to learn everything for any person (in timeless fashion), given that the person can speak absolutely fluently his own mother tongue.

This presented us with a dillema in project – what is the best middle point between putting a lot effort in teaching the necessary skills to the team members who lacked them, and to do everything twice faster without teaching any skills. In a real company, where a team is to work for much longer times than one semester, the solutions are driven by completely different arguments. Probably,

the company would use a 25:75 or 80:20 ratio between learning and doing the work.

As in our case, the solution was primarily driven by our planning – "Are we actually going to be on-time?". While this approach in itself is not wrong for a one-month project, lets see what could we have done differently.

What will be done different in next project

In the next project, which is due to start in a few month, we will benefit from:

- A notice in projects team list of the possible problems on project writing
- A well thought-out metric gathering system

There are plenty of free project management tools. It requires a great knowledge and experience to know what is important to consider as a metric and what is not. This obscurity is also in itself a risk, and thus requires that a team is capable of restructuring itself in an agile and as transparent as possible manner. This will be a great challenge for next project (and very fun challenge too).

The evaluation of our project effort can be summarized in the following diagram. It also shows the ideal effor distribution we will strive to achieve for the next project.

2.2.5 Project Risks

The main project risk (programming related risks are dealt separately) can be identified by a single source – the communication. The problems might occur anywhere throughout the project. That might be the dissatisfaction of any kind starting from the course of the project to the main theme investigation in the project report.

1. ***Not efficient communication***

   Problem: Communication is not taking place

   Severity: High

   Probability: High

   Possible solution: Talk in the team what are the problems and try to solve them. Make a "Team building session".

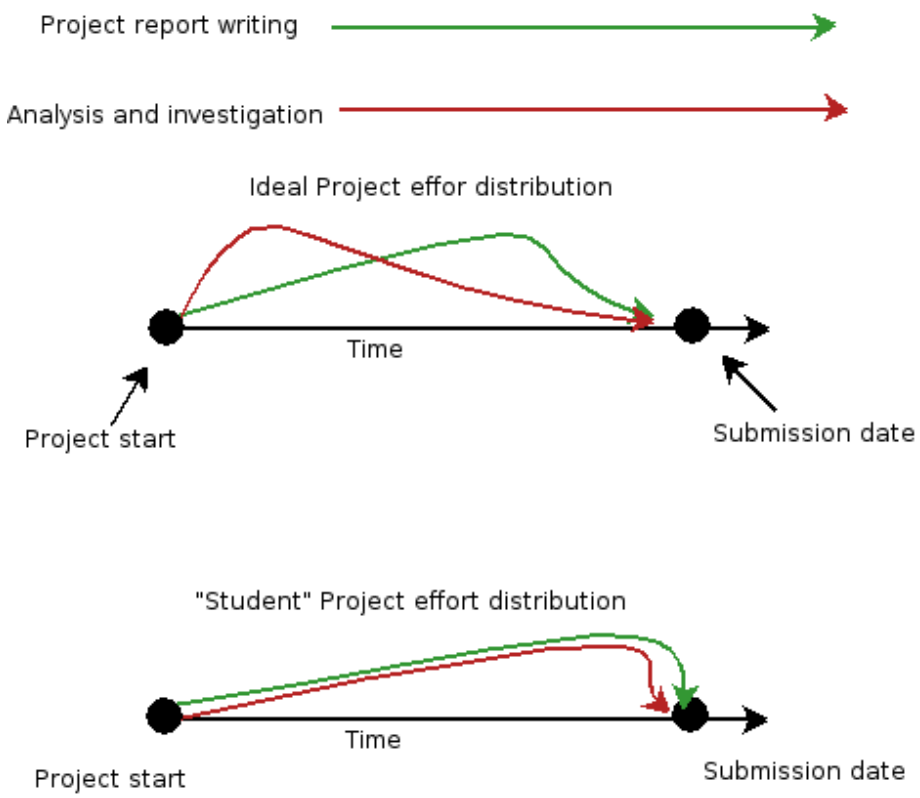


Figure 2.2: Ideal and Our Project effort distribution

2. ***Does the project has enough people to meet the objectives?***

   Problem: The project report may suffer quality damage

   Severity: High

   Probability: Normal

   Possible solution: Reduce objectives. Ensure quality and time objectives are met.

3. ***Will the people be able to accept and use so much new knowledge?***

   Problem: The project report may suffer quality damage

   Severity: High

   Probability: High

   Possible solution: Reduce objectives to ensure proper quality and time.

4. ***Is the project diverging from official (i.e. school) requirements?***

   Problem: The project may not satisfy necessary requirements

   Severity: High

   Probability: normal

   Possible solution: Seek guidance from project advisor. Make a project review and restructuring session.

5. ***Is the project schedule realistic?***

   Problem: The work still to be done accumulates in the end of the time box.

   Severity: High

   Possible solution: Make a project restructuring session, and try divide the work equally, though still it will form in big lumps. Ensure quality does not suffer.

6. ***Do all members understand requirements well?***

   Problem: The project focus not on the needed requirements.

   Severity: Low

   Probability: Normal

   Possible solution: Use acceptance strategy – try to live with this. Give people appropriate jobs (according to competence).

7.

   Problem:

   Severity:

   Possible solution:

For analysis of system risks, i.e. those arising from programming, system architecture etc, see section 6.2 System Risks on page 67 .

Chapter 3

Software Licensing

Figure 3.1: The Horse and Free Spirit

3.1 Introduction

To answer the question "How to make commercial free software" we must know what is free software. As we will see later, the freedoms of software are protected by the copyright and license mechanisms. Thus, in this chapter we will try to

answer these questions:

- What is the ***Free and Open Source software***
- What are the ***different licenses***

3.2 Free Software Definition

"Free software" is a matter of liberty, not price. The definition of free software is published by the Free Software Foundation (FSF), and can be found at their web-site[1] . FSF defines free software as four kinds of freedom, for the users of the software:

1. The freedom to run the program, for any purpose.
2. The freedom to study how the program works, and adapt it to your needs. Access to the source code is a precondition for this.
3. The freedom to redistribute copies so you can help your neighbor.
4. The freedom to improve the program, and release your improvements to the public, so that the whole community benefits. Access to the source code is a precondition for this.

Free software is not of monetarial issue. Free should be understood in terms of liberty.

3.2.1 Open Source Definition

Open source is almost the same as free software. However, both are not identical. The definition of open source is maintained by OSI – an Open Source Initiative. It is a non-profit corporation dedicated to managing and promoting the Open Source Definition for the good of the community.

A, Here is a shortened extract from the Open Source Definition:

"Open source does not just mean access to the source code. The distribution terms of open-source software must comply with the following criteria":

[1] http://www.fsf.org/licensing/essays/free-sw.html

1 Free Redistribution

2 Source Code

The program must include source code, and must allow distribution in source code as well as compiled form.

3 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.

5 No Discrimination Against Persons or Groups

6 No Discrimination Against Fields of Endeavor

7 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.

3.2.2 Differences Between OS and FS Software

So what is the actual difference between the Free Software and Open Source software?

Free Software is the idea that users of software should have the power to modify software to suit their needs.

Open Source is might be viewed as the method behind many Free Software projects. The idea is that the source code is open and available to the public, *during the development process*. This means that anyone can contribute, bugs can be found quicker, and development is spread over hundreds or thousands of people instead of just a handful. An Open Source project does not have to be Free Software, but it is effectively impossible to have a Free Software project that is not Open Source.

Both approaches have a problem arising from the same origin – their names do not convey exactly what the approaches are up to. Free can be understood as gratis. Open source can be understood as pin pointing that the source code

is the most important matter. Nor gratis, nor the source code is the essence, merely a consequence.

For more info look at http://matt.waggoner.com/freeopen.html .

3.2.3 Free Software Advantages

3.3 Legal Ways To Secure Software Freedoms

Intellectual property (IP) refers to a legal entitlement which sometimes attaches to the expressed form of an idea or of other intangible subject matter[2] . In general terms this legal entitlement sometimes enables its holder to exercise exclusive control over the use of the IP. In contrast to the tangible property, IP refers to matters produced by intellectual activity. Thus intellectual property secures such works as movies, musical works, paintings, photographs and, of course, computer programs.

The way the IP secures these works vary from one to another form. When a person produces a work, in order to have a strategic business advantage of having an exclusive right to produce copies, the material has to be protected. This protection of intellectual property comes in many forms – either copyrights, patents, trademarks or trade secrets.

Copyright secures to its holder the exclusive right to reproduce the work for a define, yet extendable, period of time. In order to apply a copyright on the work, nowadays the author does not have to do anything. After countries adopted [3] for the Protection of Literary and Artistic Works, the recognition of copyrights between sovereign countries has been established.

Nowadays, since almost all nations are members of World Trade Organization, the TRIPs Agreement requires non-members to accept almost all of the conditions of the Berne Convention, effectively making the copyright are very reliable way of securing exclusive rights.

So looking at overall picture of our system we see that whenever we create something, it is automatically copyrighted (and also as from 1989, no inclusion

[2] http://en.wikipedia.org/wiki/Copyright

[3] http://en.wikipedia.org/wiki/Berne_Convention_for_the_Protection_of_Literary_and_Artistic_Works

of copyright notice is required). This means that the use of our source code is automatically restricted to other people.

Usually, the copyright holder uses an agreement to give permissions to use the program to the user. This "agreement" is called "license". This way, the people's rights, who want to use our code, at first are restricted by copyright, and then re-granted by issuing a proper license.

So if we want to make commercial free software, we must license it under free software license.

3.4 Software Licenses

All of the following licenses further discussed falls under free-software category (except "non-free licenses"). This means that all programs must be licensed at no charge to anyone.

On the other hand, this does not prohibit of selling the software in other ways, or conveying a business model completely not focusing on revenues from the software itself. However, let's focus at the moment on examining the licenses.

Some of the material was taken from:
http://www.hecker.org/writings/setting-up-shop.

3.4.1 Public Domain

To put a software into Public Domain means that literally to give-up any copyrights held by the author.

It grants users the most freedom – from using it in any way and any where, to even exercising a right to re-release a software under proprietary license. Because of this potential problem, most open-source and free-software advocates recommend not making software public-domain; even developers who do not believe in the concept of "intellectual property" still advocate using the mechanism of copyright.

3.4.2 BSD License and Other BSD - Style Licenses

The BSD License was originally used by the University of California at Berkeley. "BSD" stands for "Berkeley Software Distribution".

The BSD License has the following main features:

- An explicit grant of the right to unlimited use in source or binary form
- A requirement that the developers' copyright notices (and related material) be retained
- Legal boilerplate to limit the developers' liability
- A requirement that the developers be credited in "advertising material".

The BSD License and licenses adapted from or similar to it have been used for several other open-source projects, such as FreeBSD, NetBSD, and OpenBSD (Unix-based operating systems), Apache (a web server) and XFree86 (an implementation of the X Window System), and others.

From the point of view of a commercial software company BSD-style licenses contain the minimum terms and conditions that an open-source license would need to have in order to be an effective license at all.

From the point of view of open-source developers, BSD-style licenses allow the maximum freedom in using the source to create derivative works. This includes the freedom to take open-source software under a BSD-style license and use it to create a proprietary product for which source code is not made available. As a result many open-source advocates recommend not using BSD-style licenses, preferring licenses that require (to a greater or lesser degree) that derivative works of free software would also be made available as free software (such as GNU GPL).

3.4.3 The GNU GPL

The GNU GPL license can be found at FSF Site[4] . The license is based on several ground issues – firstly, the software licensed under GPL is guaranteed to be free, and secondly – the software is copylefted.

We have already discussed what it means to make a software free.

Copyleft 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. Copyleft says that anyone who redistributes the software, with or without

[4]http://www.gnu.org/licenses/gpl.html

changes, must pass along the freedom to further copy and change it. Copyleft guarantees that every user has all the freedoms of those defined in the free software definition.

This license is said to be "tainting". This means that work, which has a code released under GPL license (GPL'ed code), must also be free. This is very important and must be tackled with great care. The copyleft mechanisms creates problems (and deliberately so) for proprietary software vendors who wish to build their own non-free products using GPL-ed code. For contrast, BSD is a non-copyleft license and does not cause such problems.

For example if a company has two products, and decides to use an open source business model, i.e. release one program under open source license, GPL probably would not work well, if the two programs share a substantial code base.

Sometimes it also causes many commercial organizations (who develop non-free software) to shy away from using GPL-ed software.

This case has happened to Netscape, when they released Netscape Communicator under open source license. Instead of using the GPL license (which would "taint" all their products as free software and thus make attempts to use their business model ruined), they created a new license – the Mozzilla Public License (MPL).

When the identified parts of a program can be reasonably considered independent and separate works in themselves, then the GPL license and its terms do not apply to those sections, when are distributed as separate works. If the sections are distributed as a whole, the terms do apply.

3.4.4 The Lesser GNU General Public License

The full license can be found here at FSF site[5] .

The GNU LGPL license acts primarily with the same attitude as GNU GPL license – it permits the programs freedom and forces to use copyleft, i.e. not re-release software in any non-free license. However, LGPL, as the word "lesser" implies, acts in more "loosely" – it does not "taint" code.

When a program is linked with a library, whether statically or using a shared

[5]http://www.gnu.org/licenses/lgpl.html

library, the combination of the two is legally speaking a combined work, a derivative of the original library. The ordinary General Public License permits such linking only if the entire combination fits its criteria of freedom. The Lesser General Public License permits more lax criteria for linking other code with the library. This is generally useful for software libraries when it is necessary for them to become popular.

So, in effect this license would allow proprietary software to re-use our code, but would not allow anyone to make it non-free.

3.4.5 Mozilla Public License

The MPL license is notable as an open-source license created by a commercial software company. MPL was influenced by and to some extent incorporates features from a number of other licenses, including the GPL and LGPL.

However, it introduces some new concepts. The derivative works are considered to be modification of the original source files or new source files incorporating extracts from the original source files. This is different from the GPL license, where a derivative work as a whole must be published under the same GPL license.

However, unlike the GPL license, the MPL license permits linking MPL code with proprietary code to create a proprietary program. Opposed to GNU GPL, the separation between programs is made on source file level. Of course, the other license must be compatible to the original license. This means the GNU GPL can not be merged together with MPL, since GPL license is “tainting”.

3.5 Non-free Licenses

All non-free licenses lacks essential freedoms. For example “Sun Community Source License” is not a free software license; it lacks essential freedoms such as publication of modified versions.

Proprietary licenses limit users ability to use the software, primarily by charging a fee for a license. In many cases they do not provide code availability, and where the code is available, the modifications are not allowed.

Figure 3.2: Software categories

3.6 Software Categories

Software categories summarizes our not so broad licensing investigation. The diagram below concisely and clearly shows the various categories of software.

Free Software

GPL and LGPL are free software licenses.

Open Source

If the software is termed to be open source, then in many cases it is the same as free software.

Public Domain

If the software is in public domain, basically it means that nobody holds copyright on it. If the source is also put into public domain, it also means, that whomever may later redistribute software in completely non-free manner.

CopyLefted

Copylefted means that the license can not restrict the software to be non-free. GNU GPL is copylefted license. Other licenses can be free, but not necessarily copylefted. X11 License is a good example.

GPL'ed

GPL covered licenses are compatible with GNU GPL. X11 is free software license, non-copyleft, but compatible with GNU GPL. Compatible means that the softwares released under GPL covered and under X11 licenses can be put together.

Semi-free

Semi-free software is software that is not free, but comes with permission for individuals to use, copy, distribute, and modify (including distribution of modified

versions) for non-profit purposes.

Proprietary

Proprietary software is software that is not free or semi-free. Its use, redistribution or modification is prohibited, or requires you to ask for permission, or is restricted so much that you effectively can't do it freely.

Freeware

It is commonly used for packages which permit redistribution but not modification (and their source code is not available). These packages are not free software.

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 because for most shareware the source code is not available, so it is impossible to modify the program. Also, shareware does not come with permission to make a copy and install it without paying a license fee.

Commercial Software

Commercial software is software being developed by a business which aims to make money from the use of the software. Most commercial software is proprietary, but there is commercial free software, and there is non-commercial non-free software.

3.7 Conclusion on Licensing

Till now we have answered what is Free and Open Source software. We also know what are licenses, and that as we choose some particular free software license, we impose some restrictions (peculiarities) on our possible business model.

Choosing a free license means to resign the right to ask customers to pay for their right to use a program. However, a license, which makes a source code

open, allows us to employ a completely different software development model, as opposed to traditional ways. And this, the Open Source development model, will be the focus of our next chapter.

Chapter 4

"Open Source Development"

Figure 4.1: Horse

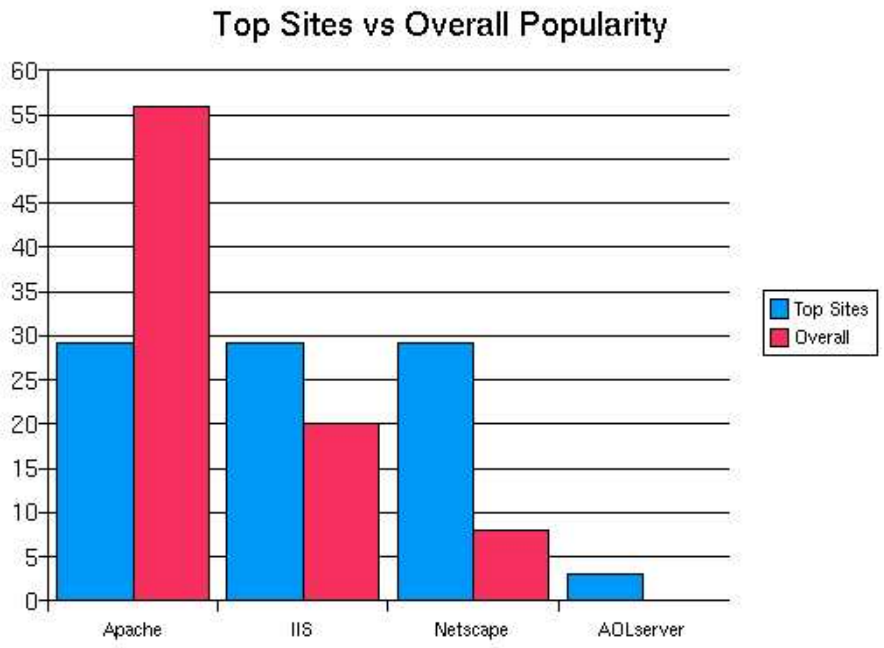
4.1 Introduction

Releasing software under an open source license does not help it directly to be a successful commercial software. Much more has to be anticipated to make it happen so. Open source licenses enables us to access such features such as open source software development. Thus, in this chapter we discuss the following problems

- What is an Open Source Development (OSD)?
- What are the reasons for and is actually OSD effective?
- What are the possibilities and consequences of using OSD for our system?

4.2 Free Software Quality

So can fee software compete with proprietary software in terms of quality, robustness, update interval and so on? Can free software be used as mission critical systems? Lets look at the web server and server operating system usage statistics.



This diagram (taken from www.e-ginner.com[1]) shows the web server popularity among the top-sites and overall popularity. The fact, that the Apache server is used at the same level as Netscape and Microsofts' IIS shows that the open source software is able to do mission-critical tasks. The Apache's popularity among overall sites is more than fifty percent, which makes it the most widely used web server.

Popularity is very important – the more people use the software, the more people get used to it. For example Microsoft knows that very well and does that on wide scale already. This company offers exclusive offerings to educational institutions, more commonly promoting the operating system and the office programs package. This gives them a big advantage (for ex. a "lock-in" effect).

Apache is praised for it's robustness and stability. The last severe bug was discovered at least a few years ago, whereas proprietary software (such as Microsoft IIS server) can not proud the same. As another advantage is Apache's vast feature list. Furthermore, some products directly reuse Apache's services, such Zope, a python based www server, reuses HTTPS protocol realization.

On top of this, Apache is an free-software product and is licensed under a GPL-compatible license. This effectively means whomever has an interest is capable on influencing the project. This can not be said for non-free software, which is an advantage of free software.

4.3 Free Software Projects

So open source software can be stable and even serve mission-critical duties. But what is that pixie dust, which makes companies developing free software capable to compete with such big competitors even as Microsoft and Sun Microsystems, Netscape? Lets look and examine the Apache and Ogre3D projects as examples of successful free software projects.

[1] http://www.e-gineer.com/articles/web-servers-used-at-the-top-sites.phtml

4.3.1 The Apache Software Foundation

Apache web server is developed by a non-profit Apache Software Foundation (ASF). It was formed in 1999 by a group of people who developed Apache already before 1995. The birth of Apache was when user community undertook the HTTPD web server written by the NCSA[2].

The web server Apache server through year 1995-1999 became leader of the market and the projects grew bigger and bigger, the need for a more coherent and structured organization that would shield individuals from potential legal attacks felt more and more necessary.

The roots of ASF were little patches that were sent between users of the web server. Suggestions in the mailing lists grew into bigger contributions. When the group felt that the person had "earned" the merit to be part of the development community, they granted direct access to the code repository, thus increasing the group and increasing the ability of the group to develop the program, and to maintain and develop it more effectively. They called it the "meritocracy" principle: literally, the government of merit.

Newcomers in the ASF are treated as volunteers, not as newcomers who want to undertake the position. There are five roles in the ASF – user, developer, commiter, Project Management Committee member and ASF member. The project grew in four years from 200 commiteers to 800 and have contributed to the market several successful projects, which are leaders in the market.

4.3.2 Ogre 3D

Another free software project exhibiting astounding popularity is the Ogre3D[3] project. It is a free system for developing applications with 3D graphics. The project has lived up to now for 4 years. It was established by Steve "sinbad" Streeting and now the team has total four members constantly developing and leading project.

The project is well-known and used by many game-developing teams. It has also a vast user community. The community send bug-fixes for the system, write documentation and tutorials, and enjoys continual interest in the project.

[2]The HTTPD web server was discontinued

[3]www.ogre3d.org

The two project greatly illustrate how open community can develop superior quality software. But probably we did not get more clear about how does it happen. We know it is capable of gathering hundreds of willing to help customers/co-developers, but is it the source code availability which makes the project flourish? Or is it the development method, which could be loosely termed "Open Source Development style(OSD)"? If so, does this OSD method contradicts other methodologies such as Unified Process, or eXtreme Programming?

We begin our investigation by examining "The Cathedral and the Bazzar" paper.

4.4 The Cathedral and the Bazzar

The Cathedral and the Bazaar[4] paper was one of the first popular papers looking the success of Linux, a free operating system kernel.

The paper discusses what the author refers to as "the cathedral and the bazzar" software development styles. The latter the author describes as "release early and often, delegate everything you can, be open at the point of promiscuity". The Cathedral style is on the other hand described as a way of developing software "in a closed community, crafted carefully by a skillful designers and programmers and no beta to be released before its time".

The author gives some guidelines of how to work in an open community by giving an example of his own project. The project was conducted adhering to author's own guidelines, made by observing the success of Linus Torvalds – the creator of linux kernel.

We believe in this study we might find what is essential in working with open communities and thus will help us solve the problem and give guidelines of what are the benefits of open source development.

1. "Every good work of software starts by scratching a developer's personal itch."

   Most of the open community software were in the beginning developed

[4]Richard Stallman – http://www.catb.org/~esr/writings/cathedral-bazaar/cathedral-bazaar/index.html

only for programmers own use. Only after releasing it the community gathered around the project and started to drive it (as was with Linux and PHP). As with the case of Apache Software Foundation, the web server Apache itself was not started by a single developer. The project was undertaken by the already existing user community.

This tells us we would definitely have higher possibilities to set up a better product if we already had a community of active developers (JSP Wiki exhibit this feature).

2. Good programmers know what to write. Great ones know what to rewrite (and reuse).

   "Reusability is the likelihood a segment of structured code can be used again to add new functionalities with slight or no modification" – gives us wikipedia explanation of what is reusability. Reusable code reduces implementation time, increases the likelihood that prior testing and use bugs are eliminated and localizes code modifications when a change in implementation is required.

   This further commits our choice to select an already well-established wiki engine. It is not only that most of the functionality will be already running, but it will be also actively developed and well bug-hunted.

3. "Plan to throw one away; you will, anyhow."

   This is not related only to open source style development – in fact, with any software the things might go unexpected way. Sometimes a system prototype becomes the system being developed, or a "throw-away" code intended to provide a proof of concept, becomes the code heavily used in the system. Sometimes these approaches are referred to as a "quick hack" or a "throw away code". Brian Foote identifies this as one of the anti-patterns and discusses it in his paper "Big Ball of Mud"[5] .

   This OSD property reminded us of agile systems. In fact, eXtreme Programming focuses on doing writing the simplest thing which possibly work. This is somewhat similar to plan to throw away and suggests us to compare OSD and agile systems. It might be so, that for our product we could

[5]http://www.laputan.org/mud/mud.html

run an OSD, whilst our team would work in agile manner. Agile methods are discussed later on page 40.

4. "When you lose interest in a program, your last duty to it is to hand it off to a competent successor."

   Open source projects usually are driven by a few devoted developers, while customers (users) have access to source code, and rightfulness to find and fix bugs or make improvements. If the main contributors leave, the community will undertake and continue the project. This is a significant advantage over proprietary software, where the project would be simply put out of production and users would be left alone.

5. "Treating your users as co-developers is your least-hassle route to rapid code improvement and effective debugging."

   This is probably one of the most crucial points – many development methodologies advice to release often and get user feedback early, but no formal methodology extend it to such extreme that customers are treated as even co-developers. Formal methodology assumptions on releasing software when the "time comes" is probably based on assumption that users want bug-free software and thus customer community should see only the final good and perfectly working result.

   No software is released bug-free and in open-source community users willingly join the bug-hunting (and even bug-killing) process and help make the software bug-free.

   The formal methodologies do not utilize the hundreds of testers, whom are willing to help, in return expecting only good software.

   Does OSD conflict with other methodologies? Perhaps no, since not every project is suitable for OSD. Not every project will have a vast willing to join user community.

   As for us, rapid code improvement and effective debugging is a valuable feature.

6. "Release early. Release often. And listen to your customers."

The concept is closely related to treating your customers as co-developers. To support this activity, open source community greatly utilizes bug-reporting (defect tracking) software. One example might be Bugzilla[6] , which is also a free software. There are many proprietary defect tracking systems, but even despite large licensing fees, Bugzilla has many features counterparts lack. Total almost 400 companies use Bugzilla, among them one can find well known Id Software, NASA and even IBM.

Indeed users are probably one of most valuable resources in open community. Here are some more tips from the author, relating to treating customers as co-developers:

* "Given a large enough beta-tester and co-developer base, almost every problem will be characterized quickly and the fix obvious to someone."
* If you treat your beta-testers as if they're your most valuable resource, they will respond by becoming your most valuable resource.
* The next best thing to having good ideas is recognizing good ideas from your users. Sometimes the latter is better.
* Provided the development coordinator has a medium at least as good as the Internet, and knows how to lead without coercion, many heads are inevitably better than one

So it is not the open source which makes the Open Source Development effective. It is the user community, which the most important aspect. Release often, have a lot of co-developers. Actually, have a lot of users, and then treat your users as your valuable resources.

So if we want to utilize OSD given advantages, we must make sure our co-developers are willing to do so. We have to create an appropriate environment. Perhaps the users must feel they have a lever, they can use when needed. If no lever exist, then the users will not respond. Furthermore, to make this possible, we must choose an appropriate license. And on top of this, if we want to make commercial software, business model must be selected carefully. It will also reflect upon organization structure and activities.

[6] www.bugzilla.org

The paper finishes citing one of the resigning principals from the open source project "Mozzilla". This project was established after Netscape announced in the year 1998 plans to give away the source for Netscape Communicator. The resigning principal announced that "Open source is not magic pixie dust". This only assures us what we concluded after our investigation of Free Software and various licenses – an Open Source project does not have to be Free Software, but it is effectively impossible to have a Free Software project that is not Open Source.

4.5 Agile Processes

While going trough Richard Stallman's paper we noted of some similarity between OSD and agile processes. We would have not peered more into these if not the need to manage our own team. To support an Open Source Development is something which is not covered by formal methodologies.

4.5.1 Agile Manifesto

"Agile" is a loose term. In the most loose fashion (and probably in the most wrong) one can characterize agile process as a lightweight process. Lightweight process poses some assumptions or values upon what the developers are doing and how they do it. It would be incorrect to think about agile software development as if it tries to convey that it is all about adopting the right process. It is welcome to understand "adopting *the right* process" in the sense that every project has different context and thus different way of solving problems. It would be not good-enough to think as if one methodology, one way of thinking is at least good for any project.

The term agile started to spread in early nineties, and the AgileAlliance was formed in 2001 to promote the concepts of agile software development, and help organizations adopt them. "Manifesto for Agile Software Development"[7] was issued by the AgileAlliance, which propose these values:

- ***Individuals and interactions*** over processes and tools

[7] http://www.agilealliance.org/

- ***Working software*** over comprehensive documentation
- ***Customer collaboration*** over contract negotiation
- ***Responding to change*** over following a plan.

Furthermore, in the manifesto is stated that "while there is value in the items on the right, we value the items on the left more". We believe, that the latter note is very important since it stresses the importance of context. I.e. that both sides are important, and to neglect means to mis-understand and mis-perceive. However, the stress should be put on the left-sided terms.

4.5.2 Agile Values

Agile software development can be also viewed as a collection of values and principles, taken from "Principles behind the Agile Manifesto"[8] .

- Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
- Welcome changing requirements, even late in development.
  Agile processes harness change for the customer's competitive advantage.
- Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
- Working software is the primary measure of progress.
- Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
- Continuous attention to technical excellence and good design enhances agility.
- Simplicity – the art of maximizing the amount of work not done – is essential.
- The best architectures, requirements, and designs emerge from self-organizing teams.

[8] http://www.agilemanifesto.org/principles.html

- At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

The list is a little shortened That is a long list, but it is all uncut, so the reader can make his own opinion.

4.6 Similarities Between Open Source and Agile

Indeed, many points mentioned above relate to the open source development. Point saying to release software early and constantly relates to "Release early. Release often. And listen to your customers.". Building projects around well motivated people is nothing different than simply scratching one's personal itch. Technical excellence and good design is a consequence (unfortunately only in the best case) of treating your users as co-developers. And the last one – The best architectures, requirements, and designs emerge from self-organizing teams – is just an exact description of what happened with Apache web server.

There were many sessions to relate agile to open source. One of them is "BoF session on Open Source and Agile at Linux Bangalore 2004"[9] , where investigation on the philosophy behind open source and agile was conducted. The participants agreed most of the XP practices such as SmallReleases, SimpleDesign, Testing, Continuous Integration, ReFactoring and others fit well with OSD. The values of XP for communication, simplicity, feedback and courage are the same in both approaches.

What this shows to us is a perfect correlation of Agile processes and the Open Source Development. Acting upon the same principals and guidelines we will strengthen our focus motivating both the users and the team.

4.7 Agile Methods

Agile approach is only a set of principles and values, but it is not a concrete methodology. Put principles and philosophy together, and we get various ways to build software and/or organize teams. This section describes some of the

[9] http://c2.com/cgi/wiki?OpenSourceAsAgileProcess

agile methods, which we could use in relation to our open source development activities.

1. ***AD – Agile Data Techniques***

   Agile Data techniques is all about designing data in an agile manner. Using this technique, the data in the software is viewed as the most important part. Focus is on relational databases, domain modeling, differences between data models and object models, object orientation with data and working with legacy data. The technique was founded by Scott W. Ambler and can be found at http://www.agiledata.org/.

2. ***AM – Agile Modeling***[10]

   Agile Modeling website gives a very good explanation of what it is – "Agile Modeling (AM) is a collection of values, principles, and practices for modeling software that can be applied on a software development project in an effective and light-weight manner. Agile models are more effective than traditional models because they are just barely good enough, they don't have to be perfect. You may take an agile modeling approach to requirements, analysis, architecture, and design.".

   They also say, that AM is a supplement for existing methodologies and that it is more of an attitude, not a prescriptive process.

   Interestingly, we find it very similar to the eXtreme Programming methodology. To further surprise, we found an article "Agile Modeling and the Rational Unified Process (RUP)". Interesting is the fact, that our team half a year before already already did practically the same paper.

   Our article (named drastically "XP vs UP") tries to blend the eXtreme Programming and Unified Process methodologies and looks what each can offer to the other.

   The AM and RUP paper looks how the agile philosophy and practices fit the RUP methodology, and conclude that the biggest obstacle for agile modeling is not the methodology itself, but the culture.

[10] http://www.agilemodeling.com/

Slightly differently, but not contradictory is our articles conclude stating, that it is completely possible and even advisable (were appropriate) to blend agile eXtreme Programming methodology with UP.

3. ***ASD – Adaptive Software Development***[11]

   "The practices of ASD are driven by a belief in continuous adaptation – a different philosophy and a different life cycle-geared to accepting continuous change as the norm. In ASD, the static plan-design-build life cycle is replaced by a dynamic Speculate-Collaborate-Learn life cycle. It is a life cycle dedicated to continuous learning and oriented to change, reevaluation, peering into an uncertain future, and intense collaboration among developers, management, and customers."

4. ***Crystal***[12]

   Crystal is a family of human-powered and adaptive, ultralight, "shrink-to-fit" software development methodologies.

   "Human-powered" means that the focus is on achieving project success through enhancing the work of the people involved (other methodologies might be process-centric, or architecture-centric, or tool-centric, but Crystal is people-centric). "Ultralight" means that for whatever the project size and priorities, a Crystal-family methodology for the project will work to reduce the paperwork, overhead and bureaucracy to the least that is practical for the parameters of that project.

   "Shrink-to-fit" means that you start with something possibly small enough, and work to make it smaller and better fitting.

   Crystal is non-jealous, meaning that a Crystal methodology permits substitution of similar elements from other methodologies.

   Crystal collects together self-adapting family of "shrink-to-fit," human-powered software development methodologies based on these understandings:

   (a) Every project needs a slightly different set of policies and conventions, or methodology.

[11] http://www.jimhighsmith.com/
[12] http://alistair.cockburn.us/crystal/

(b) The workings of the project are sensitive to people issues, and improve as the people issues improve, individuals get better, and their teamwork gets better.

(c) Better communications and frequent deliveries communication reduce the need for intermediate work products.

5. ***Scrum*** Scrum is wrapping existing engineering practices, such as eXtreme Programming and RUP. It is an agile process which bases on two ideas:

   (a) ***Team empowerment*** : Once teams are given work to do, they are responsible for figuring out how to do it. The team does the best it can during each increment. While a team works, their only interaction with management is to tell management what is getting in their way and needs to be removed to improve their productivity.

   (b) ***Adaptability*** : Scrum uses so called "punctuated equilibrium". This means the team has a list, which can not be changed from outside (i.e. only the team has control of it). The list is reviewed every some time interval (e.g. thirty days), evaluated and changed. The decision is on what the team has accomplished and what the environment dictates is the next most important thing to do.

6. ***TDD – Test-Driven Design***[13]

   Test driven design, as its name implies, is all about driving code and design with tests.

   (a) Quickly add a test, just enough code to fail.

   (b) Run your tests and ensure the new test does in fact fail.

   (c) Update functional code to make it pass the new tests.

   (d) Run tests again. If they fail, update functional code and retest.

   (e) Remove duplication, extract common setup into the fixture, refactor more if needed.

   The above sequence is repeated continuously throughout the programming process. Each cycle has a duration of minutes, if you're not capable to fix

[13] http://www.testdriven.com

a failing test within minutes, then throw away the test and the code that you wrote to satisfy the test and design a simpler test.

7. ***XBreed***[14]

   XBreed is a combination of Scrum and XP, which is very natural: Scrum provides a solid management framework, while XP provides a basic but complete set of engineering practices. The result is a lean but very mean (very effective) way to run software projects.

8. ***XP – eXtreme Programming***

   eXtreme Programming is a discipline of software development based on values of simplicity, communication, feedback, and courage. It works by bringing the whole team together in the presence of simple practices, with enough feedback to enable the team to see where they are and to tune the practices to their unique situation.

   While other methodologies are also doing refactoring (re-designing the code to make it more efficient) XP pushes it to extreme making refactoring a day-to-day activity, and putting design on continual development and evolution.

4.8 Conclusion

Showing a source code is a great convenience and a great power which we will take advantage of. By selecting a license, which enables software source code viewing and modification, we enable ourselves using an Open Source Development.

As we have seen, there exist strong free software projects, and user community is one of the key things. However, making a source code public is not a solution, it merely enables us to experience advantages of customer acting as our most valuable resources. This in effect greatly affects our activities and organization.

Firstly, we must adapt our team to embrace Open Source development. This in effect means we must ensure our own development style is in resonance with

[14] http://www.xbreed.net/index.html

our (still future) communities pace and direction.

Secondly, our business model as well must adapt to community accordingly.

As for the first issue, the solution was proposed already in project start chapter – XBreed methodology proposes agile manner both in team management, and software development. The tools we use are also very important. Bugzilla could be used as a robust and reliable bug-reporting tool. SVN tool (source version control, also used in writing this report and creating a system, developed on top of CVS) will help us synchronize with the communities pace – convenient code repository will be one of the important tasks.

Of course, this analysis of our adaptation is very trivial and too small. We must firstly analyse all possible business models, evaluate our decision to choose a free software, not proprietary; understand our advantages and disadvantages; and the market.

Overview on page 90 takes a glance on the importance of organization structure.

Chapter 5

Business Models

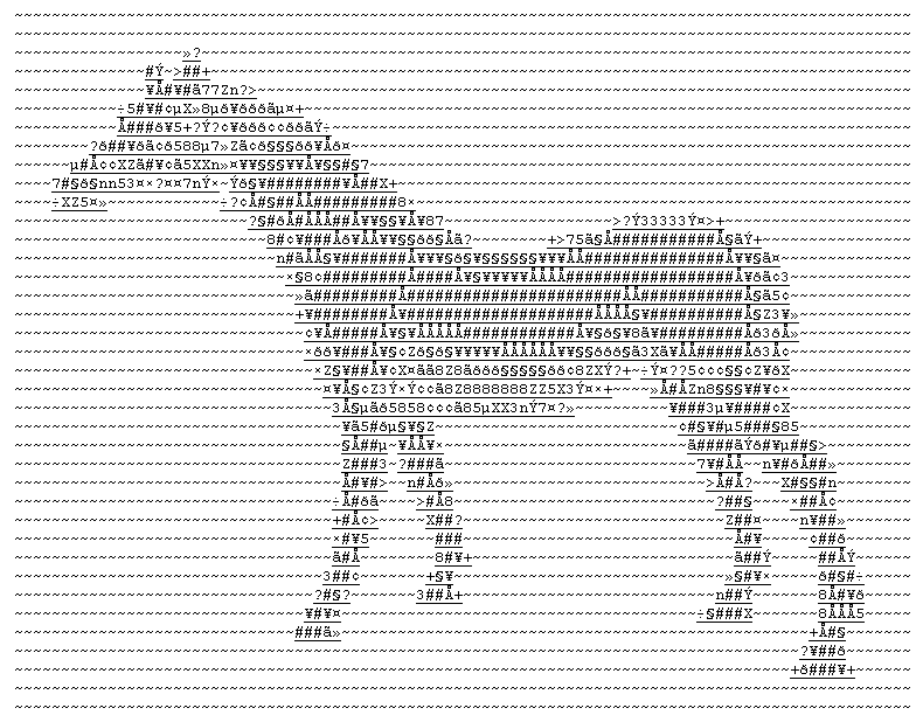


Figure 5.1: Horse

The next step to answer the question “How to make commercial free software” after understanding what is free software and what advantages we gain (in particular - the Open Source Development) is to analyse the possible business models.

A business model is a strategy to achieve goals. It is a prescription of how to get a profit for a business. And since business are more or less in most cases after the profit, a decent business model is a must have option.

Business model needs a comprehensive attention, since it involves both strategy and implementation, and many other things must be evaluated.

So the goal of this chapter is to discuss what must be taken into account when choosing a business model for our company and to suggest the best one.

In order to do that we will discuss:

- What is Business Model
- Current situation in the market (demand and offering)
- Can we offer anything unique
- What licensing model suits our product best
- What Business Model(s) we can choose

5.1 What is Business Model

A business model (also called a business design) is the mechanism by which a business intends to generate revenues and profit. Business model consist of the following components:

- Customers
- Competitors
- Offering
- Activities and organization
- Resources
- Factors and production in suppliers
- Managerial and organizational, longitudinal process component

When analysing a business model, you can see it as a summary of how a company plans to serve its customers. It involves both strategy and implementation, and is the totality of:

- How it will select its customers – advertising new products (media include billboards, street furniture components, printed fliers, radio, cinema and television ads, web banners, Web Popups, bus stop benches, magazines, newspapers, town criers, sides of buses, taxicab doors and roof mounts); customers are workers in company; company look for customers by itself.
- How it defines and differentiates its product offerings – is the modification of a product to make it more attractive to the target market by differentiating it from competitors' products.
- How it creates utility for its customers – Companies create the best that they can, and make the better opportunities than their competitors, because they are interesting to recruit more new costumers. The customers can choose between two or more opportunities and select the one, which makes more utility. For example: "you can observe me preferring coffee instead of tea – coffee gives me a higher utility than tea."
- How it acquires and keeps customers
- How it goes to the market (promotion strategy – involves disseminating information about a product, product line, brand, or company; and distribution strategy – deals with logistics: how to get the product or service to the customer.)
- How it defines the tasks to be performed
- How it configures its resources
- How it captures profit – make best products and offerings

5.2 Market Analysis

A newcomer in business must be unique somehow, otherwise it will not be able to get market's share. There are many different ways to attract customers:

Price Leadership – A company can offer cheaper products, but it must be aware that it can lead to lower revenues

Closer to Client – You can attract customers locating business in a place more convenient for buyers

Product Differentiation – Customers can be attracted by offering products, which only differ in appearance.

Product Substitution – where customers choose between producst from the same family group, but which are not the same.

So the main purpose of this chapter is to analyse current situation in the market and find some empty niches.

Since there are a lot of competitive products in the market, it would be difficult to analyse them separately. However all the offering can be divided in two groups (proprietary and Free Software solutions), so advantages and drawbacks can be easily compared.

5.2.1 Proprietary Knowledge Management Solutions

There are many companies in the market offering good KM systems that we will have to compete with. So it is important to understand why proprietary solutions are so popular among users in order not to miss their needs. The main advantages of proprietary solutions are (from a customer's point of view):

- *Good service.* If customer pays, vendor (or other companies) can install solutions, give a support information and upgrade when needed.
- *Long-term support.* If you buy products form a bigger vendor, you can be more sure that it will not disappear one day leaving you on your own.
- *Feature-rich products.* Since there are a lot of similar products in the market, features plays a significant role and developers are trying to implement as many as possible.
- *Good integration with other (especially developed by the same company) systems.* It creates some kind of lock-in effect and gives convenience for users.

Drawbacks:

- *High price.* It is because to develop a good product costs much and invested money must be returned.
- *Software bugs.* Sometimes minor bugs are kept unfixed for a longer time.
- *Limited customisation.* Since source code exclusively belongs to a vendor, it is difficult to modify software in order to adopt to your needs.
- *Lock-in.* Because standards for holding data are not usually unveiled by a vendor, it becomes difficult to switch to another product.

5.2.2 Free Software Solutions

Not all the people were satisfied with proprietary software so Free/Open Source movement has been established. Community has developed solutions for many areas. Biggest advantages of free software are:

- *Low Price.* The software is given away.
- *Stability.* Free Software solutions are well known for their robustness and stability. It is because programmes are reviewed by many independent people in the world.
- *No Lock-in effect.* Switching costs are everywhere, but the effect is not created deliberately, it is simply not as big as in proprietary software.

Free/Open Source solutions are not used everywhere, because they have drawbacks too:

- *Poor service.* Developers are mostly concerned on the product and do not establish things like help-desk and so on. This is very unattractive for customers.
- *Feature-less programmes.* Since developers implement only things which they find useful, some important features, as considered by companies, can be missing.

5.2.3 Possible Niches in the Market

It can be seen that both proprietary and Free/Open Source solutions has advantages and disadvantages. Free Software solutions posses good quality and lower cost, but lacks good service and support which is very important in business. Usually, the service and support of software is lacked because the products are developed solely by the user community, without any commercial activity around it.

Ogre3D is an example of a successful free 3D graphics engine, where a business is giving service and support for other companies developing commercial games.

So there is a niche in the market for Free Software service and support. It does not require to invest much in order to enter the market, because products are already widely used.

We decided to improve an existing Wiki engine during the project, in order to get a good Knowledge Management system for software development companies. In this situation companies, which already use the particular Wiki system are potential clients. We will also broaden our market by improving the system and making it more attractive for other companies.

So we can earn money by making business around a good Free Software solution, making it even better.

5.3 How Lock-in Effect Works?

5.3.1 What is Lock-in Effect?

Lock-in is a situation in which a customer is dependent on a vendor for products and services and cannot move to another vendor without substantial switching costs, real and/or perceived. By the creation of these costs to the customer, lock-in favors the company (vendor) at the expense of the consumer. Lock in costs create a barrier to entry in a market that if great enough to result in an effective monopoly.

5.3.2 Where Lock-in Effect Occurs?

It is often used in the computer industry to describe the effects of a lack of compatibility between different systems. Different companies, or a single company, may create different versions of the same system architecture that cannot interoperate. Manufacturers may design their products so that replacement parts or add-on enhancements must be purchased from the same manufacturer, rather than from a third party. The purpose is to make it difficult for users to switch to competing systems.

This approach is not limited to the computer industry, however. For example, as of 2004 Sony digital cameras typically use add-in memory that can only be acquired from Sony, and this memory is typically much more expensive than alternatives available from multiple sources. Lock-in may eventually also be damaging to the company or industry in question. Sun Microsystems' unwillingness to open Java to external standardization bodies and the lack of multiple competing Java runtime implementations is widely held to be the reason Java has failed on the desktop.

One way to create artificial lock-in for items without it is to create loyalty schemes. For example, frequent flyer miles that can only be used with one airline create a perceived cost of switching airlines, as do supermarket "iscount"cards.

5.3.3 Free Software and Lock-in Effect

In the 80s and 90s, public, royalty-free standards were hailed as the best solution to vendor lock-in. The weakness of such standards was that if one software vendor achieved a dominant market share, "embrace, extend and extinguish"tactics could be used to obsolete the standard.

Since the late nineties, the use of free/open source software (FOSS) has been pushed as a stronger solution. Because FOSS software can be modified and distributed by anyone, the availability of functionality cannot tie a user to one distributor. Also, FOSS tends to cling to standards. The ineffectiveness of distributor lock-in means there's no incentive for FOSS developers to invent new data formats if usable (royalty-free) standards exist.

5.4 Business Models Comparison

Over the years, business models[1] have become much more sophisticated. Now there are many of business models. There are couple of them:

- Loyalty model
- Subscription
- Servitization of Products
- Support Sellers
- Loss Leader
- Brand Licensing
- Collective system model
- Monopolistic model etc..

We will analyse just part of them with basic idea to find out which suits us best. We also will discuss the possibility to mix business models, in order to increase the possibility of our successful entrance into the market. So let us start.

5.5 Ideas And Meanings About The Product

We have already analysed the market and decided to pay more attention on better support and service. So we will regard on this by our business model(s) choice.

Since we produce free software product and, according to the market analysis, this kind of software create just little *lock-in effect* 5.3, we must keep in mind that it would be definitely hard to "hold" our customers and be assured they will not switch to competitors. Free software does not create high switching-costs. For this reason we must pay high attention on our system buyers and try to make them satisfied, because we can deduce that our success depends very much upon our customers fulfillment.

[1] http://en.wikipedia.org/wiki/Business_model

Free software benefits and revenues occurs in the longer run. Actually, we can not get profit from "one-time" sell at all, but we may sell our product-service for particular period with expectance to prolong it more longer.

However, we must ensure good our system's work and it's support, because so we will be able to provide satisfaction for our customers and hold them. Even if the company, which use free software, decide to change supplier, then it still will has (less then proprietary company) switching-costs: retraining of employees, new system installing and run, possible work flow changes etc.

Well, it seems definetelly, it would be the best decision to choose such business model, which:

- Gives opportunity to have closer relations with customers.
- Propose long-time service and support

For this reason let us discuss the loyalty business model.

5.6 Loyalty Model

According to Wikipedia[2] , the loyalty business model is a business model used in strategic management in which company resources are employed so as to increase the loyalty of customers and other stakeholders in the expectation that corporate objectives will be meet or surpassed. A typical example of this type of model is: quality of product or service leads to customer satisfaction, which leads to customer loyalty, which leads to profitability.

The model then looks at the strength of the business relationship and proposes that this is determined by the level of satisfaction with recent experience, overall perceptions of quality, customer commitment to the relationship, and bonds between the parties. A single disappointing experience may not significantly reduce the strength of the business relationship if the customer's overall perception of quality remains high, if there are few satisfactory alternatives, if they are committed to the relationship, and there are bonds keeping them in the relationship. The existing bonds crate a exit barrier. There are many types of bounds, like a legal bonds (contracts), technological bonds (shared technology),

[2]http://en.wikipedia.org/wiki/Loyalty_business_model

economic bonds (dependence), knowledge bonds, ideological bonds, psychological bonds and others.

The model then examines the link between relationship strength and customer loyalty. Customer loyalty is determined by three factors:

- Relationship strength
- Perceived alternatives
- Critical episodes

The relationship can terminate if: the customer moves away from the company's service area; the customer no longer has a need for the company's products or services; more suitable alternative providers become available; relationship strength has weakened; or the company handles a critical episode poorly.

Let us look how model affects profitability. The fundamental assumption of all the loyalty models is that keeping existing customers is less expensive than acquiring new ones. According to Buchanan and Gilles (1990), the increased profitability associated with customer retention efforts occurs because:

- Account maintenance costs decline as a percentage of total costs (or as a percentage of revenue)
- Long term customers tend to be less inclined to switch, and also tend to be less price sensitive
- Long term customers may initiate free word of mouth promotions and referrals
- Long term customers are more likely to purchase ancillary products and high margin supplemental products
- Customers that stay with you tend to be satisfied with the relationship and are less likely to switch to competitors, making market entry or competitors' market share gains difficult
- Regular customers tend to be less expensive to service because they are familiar with the process, require less "education", and are consistent in their order placement

- Increased customer retention and loyalty makes the employees' jobs easier and more satisfying. In turn, happy employees feeds back into better customer satisfaction in a virtuous circle

For more stated reasons look at Wikipedia[3] .

Still it is just presumption. We see this model are suitable to our product. Indeed, virtually almost every company today has a loyalty components in it's model. Well, the loyalty business model is adapt to use it jointly in mix with different one(s). There are many different business models, which we could use. So we will be able pick and even combine another business model(s) to get more profit. Let us look for another one(s).

5.7 “Traditional Business Model”

Software development companies earlier were using what we term a “Traditional Business Model”. With this model, companies charged a fee for a product license. It was usually very restrictful thus prohibiting customer to exercise some or all of the freedoms, including studying and adapting the software to his own needs.

This model has a disadvantage from the customer perspective – as soon as the vendor received money, the customer looses any chances to influence the supplier. This caused a lot of problems when the customer found software not having the necessary functions or a fault. In this case, the purchaser had no other option but to wait and buy another software-release or upgrade.

In the nineties software companies began to embrace the other model, which was called Subscription Model.

5.8 Subscription Business Model

Nowadays suppliers tend to pick business models (BM), which are based on subscription. Rather than sell products directly, more and more companies are selling monthly or yearly access to a product or service. According to Wikipedia[4]

[3] http://en.wikipedia.org/wiki/Loyalty_business_model

[4] http://en.wikipedia.org/wiki/Subscription_business_model

this, in effect, converts a one-time purchase of a product into a recurring renting of a product with additional services. We can divide this approach into two parts:

- Subscription of service
- Subscription (renting) of software

At First, to be more clear, we explain the difference between support and the service.

Difference Between Support and Service

Indeed, as far as we can understand, term "support" is part of "service" meaning. We can say even "support service", but there are couple issues. When you bay the service you can get support too, but not necessarily. It depends on company, which sells service. When you bay system support from the company you also being informed what kind of support company propose (maybe suppliers will maintain just systems updates and support for new version you will have to bay again). So both terms have no clear bounds. For this reason we will describe our terms.

In our opinion, there are a little difference between support and service terms. Support is more oriented in to recent system's bug-fixing, system's module reconfigurating or changing, "customer wish implementing", updates, new versions, training, consulting and branding. Service, on the other hand, propose an broad area of more intangible stuff like hot lines, "customer wish committing", bug reports, knowledge sharing etc., but it also involves training, consulting and branding. So now we have our description of these terms and can look further for our research.

5.8.1 Subscription of Service

Subscription of service means that customer can order some support from vendor. Actually, when this BM is used the product itself sometimes is given for offering market for free. At least, supplier state it, but he returns his expenses from fee we pay.

Subscription of service reduces the uncertainty we can face when problems with product occurs. It is useful for customer, because he/she has no need to look or even hire some product administrated employee. The product buyer just use it and do not pay much attention on it. For supplier it is useful since he/she will be sure to have revenue for longer time. Also, in many cases (such as integrated software solutions), the subscription pricing structure is designed so that the revenue stream from the recurring subscriptions is considerably greater than the revenue from simple one-time purchases. This model will be also useful in our companys' selling strategy. However, subscription of service are similar to Service Economy 5.9, but still have differents. Now let us look at next kind of subscription model.

5.8.2 Subscription of Software

Subscription of software BM is based not just only on product purchasing, but also on offered support. This business model differs from earlier mentioned, because here are more stated the renting of product and it post-sales support. Still it is beneficial in both way: customer becomes some newer versions and updates of product as soon as possible. Suppliers also will become profit from it, because their become revenue earlier and even in development state.

Now we see why this BM is useful for software development company. It gives a warranty that software providers will keep their buyers and still will have a revenue. Besides, it was made some research in this area and the results was that to keep old customers are more cheaper then to attract new ones. So, businesses benefit because they are assured a constant revenue stream.

A subscription of software model may be beneficial for the software customer if it forces the supplier to improve its product. According to Christopher Lochhead, chief marketing officer of Mercury Interactive (as cited in the CNET[5] article listed in the "References" section) idea, a psychological phenomenon may occur when a customer renews a subscription, that may not occur during a one-time transaction: before renewing a subscription, the customer may find it easy to sever his/her relationship with the supplier.

Subscription of software business model contradicts a little with free software

[5]News.com

idea, since this model states the software selling. So this part of Subscription's model do not suits our company. However, support area is good to our offering strategy.

5.8.3 Disadvantages of Subscription Model

However there are also drawbacks to subscription models. Often, as in the case of software, the customer may wish to pay a one time fee for the security of knowing that no further payment is necessary. Also subscription models increase the possibility of vendor *lock-in* 5.3 and this herewith might contradict with *Free* Software ideology, and consumers may find repeated payments to be onerous.

Finally, subscription models often require or allow the business to gather substantial amounts of information from the customer (such as magazine mailing lists) and this raises issues of privacy. For these discommodity let us discuss for more clear and earlier referred models – Service Economy [look below] and Support Sellers 5.10.

5.9 Service Economy

As we have already mentioned, usually there is used not just one pure kind of business model, but some mix of couple. Certainly if we use the subscription BM, we will offer some service to our customers. So, we can also refer to the service economy in it. According to Wikipedia[6] , service economy can refer to one or both of two recent economic developments:

- One is the increased importance of the service sector in industrialised economies
- The term is also used to refer to the relative importance of service in a product offering

That is, products today have a higher service component than in previous decades. In the management literature this is referred to as the servitization of products. Virtually every product today has a service component to it. The old

[6]http://en.wikipedia.org/wiki/Servitization_of_products_business_model

dichotomy between product and service has been replaced by a service-product continuum. This is a service-centric view of the economy: everything purchased has a significant service component.

Well, looking from loyalty business model's position at service economy we must admit it is good for our product. It also fill in the free software service-gap we found out in market analysis. However it is more economical position then model. So this subchapter is well to better understand what means service, but in general we should stay at the Subscription of Service business model.

5.10 Support Sellers

Support Sellers business model's software-related revenue comes from media distribution, branding, training, custom development, and post-sales support of software instead of from traditional software licensing fees.

Revenue is generated by selling two broad categories of items: physical goods, e.g., media and hard-copy documentation, and/or services, e.g., technical support. All open-source license types can be used with this model.

According our earlier market research, this model (with couple exceptions) also fits us. In market analysis we have found out a gap in Free Software service area. So Support Sellers model suits us as physical goods proposing. In our company case, technical support's area drops out, because we do not develop the hardware.

There are more business models, which may fit us well. Let us look them too.

5.11 Loss Leader

According to Loss Leader model (also known as "Bait and Hook Model") works by selling a "master" product at a subsidised price or even giving away, and making the profit on high margin "consumables" that are essential to the use of the master product. Free software product generates little or no revenue, but providing the product makes it more likely that customers will buy other products that are sold using the traditional software business model [remember

"Traditional Business Model" 5.7].

Generation of revenue from the free software product could be done as in the Support Sellers [mentioned earlier] model, that is by selling media and services. However typically the most of revenue would be generated through sales of other software products. Free software product could increase sales of such traditional products in various ways:

- By increasing the overall base of developers and users familiar with and loyal [look at Loyalty business model 5.6] to the vendor's total product line
- By making the traditional products more functional and useful (in essence adding value to them)
- By helping build the overall vendor brand [look next subchapter] and reputation

This model must be carefully studied, since if free software product shares common code with a proprietary software, any tainting licenses such as GPL can not be used. Less GPL, BSD-style license or Mozilla Public License fits well.

5.12 Brand Licensing

According to hecker.org[7] , in the "Brand Licensing" model a company makes the software product itself open source but retains the rights to its product trademarks and related intellectual property, and charges other companies for the right to use those trademarks in creating derivative products distributed under the exact same brand name. This of course requires that the product exist in at least two different forms with two different names: the "official" product referred to by a trademarked name, and the "unofficial" product referred to by a separate name.

Author states, that in general the two product versions (with and without associated trademarks) could be built from identical or near-identical source code; however from the perspective of the market they would be two different products with possibly different perceived value. (For example, the branded

[7] http://www.hecker.org/writings/setting-up-shop

product may have undergone additional testing and validation not done with the non-branded product.) If you convert a product to open source then a third party wishing to distribute a product based on that source (for example, for distribution as part of a special on-line service) may also wish to separately pay you for a license to use your trademarks in association with that product. The price of that license can reflect the brand value and reputation that your company (and by association, your trademarks) have in the marketplace.

This business model requires a lot of time (particularly small company, which begins it's business) to create brand. So investment return per longer period. It does not contradict very much with our selling strategy, since we spend quiet enough time to entice the customers anyway. We creating the system, which combines JSPWiki based system with documentation generation system. So it means we could use "Wiki" brand to attract customers, which use Wiki-based systems. But it would be useful until we would use our system and if we decide to change system or crate new one we would probably loose our customers.

However, using this model is not possible with traditional free software products because the product "brand names" are typically not formally registered as trademarks (in the case of Linux this caused a dispute requiring legal action to resolve). For this reason it would be better to pick Collective System Business Model [next subchapter].

5.13 Collective System Model

According to Wikipedia[8] , a collective business system is a business organization or association typically comprised of relatively large numbers of businesses, tradespersons or professionals in the same or related fields of endeavor, which pools resources, shares information or provides other benefits for their members. There are created such Collective Business Systems:

- Trade Association [look below]
- Cooperative 5.13.2
- Franchise 5.13.3

[8] http://en.wikipedia.org/wiki/Collective_business_system

Let us analyse them closer.

5.13.1 Trade Association Model

Trade associations are non-profit organizations in which the individual members are companies or individuals engaged in a common business pursuit. Competitors join together to create a platform format in which they deal with common problems of their industry. The trade association bears no credit risk in these transactions but instead, provides chosen vendors with access to a large body of member customers.

This model would be one of the ways, which would be useful to our company to enter into market. However, today the trade associations are of little help in enabling their small, independent members to compete with large national competitors.

5.13.2 Cooperative Model

A cooperative is a non-profit organization somewhat similar to a trade association. A significant difference between the cooperative and the trade association, however, is that with a trade association, the members have a non-equity position in the association, whereas in the typical cooperative the members will have an equity interest as all members of the cooperative own a portion of the cooperative

Cooperative model fits us not well, because we are just small company with little amount of value. So in this case the Trade Association Model would be better to us. And now let us look at the final business model: Franchise.

5.13.3 Franchise Model

The franchise is a for-profit collective business system wherein the franchiser offers proprietary products or services to its franchisees. The franchiser generally gives considerable marketing support to its franchisees. In exchange, the franchisees are subject to a substantial amount of control by the franchiser concerning its operations and marketing including the use of the franchiser's trade names, trademarks and copyrighted materials. There is generally a substantial

fee paid by the franchisee for the privilege of becoming a franchisee.

The franchise structure severely inhibits the independence of the franchisee. The success of the franchisee is tied to the success of the franchiser. The franchisee is not free to introduce non-approved products or services. It is also generally restricted to introducing innovative business or marketing strategies by the extensive control imposed by the franchiser.

For our Free Software orgnisation it would be difficult to use franchise model, since:

- This model is oriented more for propertary goods selling
- There are imposed burdensome fees on model's members
- Franchise it not free to innovations. For this reason it is very difficult propose new non-tested product or service to the market.

So, definitely, this model are not proper to our company.

In this chapter we have discussed many business models, so now is time to make a conclusion and pick the best business model(s) for our product.

5.14 Conclusion

Now we will decide what business model we will use. At first we must remember market analysis results. They state there is a gap in market for Free Software – *service and support* areas. We will fill those in. Well, looking at mentioned business models, we see Subscription of Service model suits us perfectly. According to this model we will transform our product into product-service and will supply for users additional services they lack, such as free Softvare call centers, bug reports etc.. For the same reason our company also can use Support Sellers model (with a little changes). With this we would provide physical support like media, consulting, training and others.

The second thing is *comunity*. Free Softwere depends much on people, which are working not in company (it is like a kind of outsourcing). We must pay high attention on them and sure they will stay and will not switch to competitors. So for our system we create is important to use Loyality Business Model. We will benefit, because it is cheaper to hold old customers than acquire new ones.

There are many similar products on the market and it is very important to show for buyers as many unique points of produced product as possible. For this reason we will try to expose our developed system with many useful features. We have to try do this also in different way than our competitors do. Our unique points will be service and support of Free Software and high atention to customers loyalty to our company.

There is one more issue – we just begin entering into the market and *it takes time* to lure customers. So we should pick flexible offering strategy to be able to adapt our product for customers' needs as much as possible. For this reason it would be good to choose a system, which would help us faster change our resources and offerings to achieve highest benefit possible.

Chapter 6

System

Figure 6.1: Horse

Our project is based on two ground assumptions. First, we have a problem of how to make commercial free software. A second issue is that we have already a vision (see section 1.4.1).

This chapter will focus on describing the process of our systems development.

First of all requirements and possible risks will be listed. Possible solutions for the risks will be discused and the system design will be made.

6.1 Requirements

Since our aim is to combine Knowledge Management and Source Documentation systems, requirements are mostly derived from them.

6.1.1 Functional Requirements

6.1.2 Non Functional Requirements

Communication protocols with external programmes, structures of any existing data and other relevant for the project standarts external are described in this section.

1. ***HTML usage.*** Since all the data will be presented using browsers, programme must provide an output which complies with HTML standarts.

2. ***Parse Source of most programming languages.*** It is impossible to make parsers for many programming languages during project period, but the system architecture should be easily extendable for new parsing ones. At least one parser must be present for demonstration purposes.

6.2 System Risks

Most projects have risks which must be tracked in order to succeed. Sometimes people are trying not to think about possible problems or don't bother theirself with searching for a solution as soon as possible. This atitude is not good, because sometimes a single problem can require much time for fixing or even fail the project if it is not solved at the right time.

So we will try to list all relevant risks for our project in this section.

1. ***Source Parsing is Complex.***

   Problem: Source parsing is very tricky job, because all the rules of a language must be known, taken to account and programmed propertly.

Severity: high

Probability: high

Possible solution: There are allready written open source parsers on the internet and maybe some of them can be used.

2. ***Source Parsing Time.***

   Problem: Even with good parsers it takes time to analyze a single file. In our situation there can be hundreds of files, so a server would not be capable to serve even a few queries at a time.

   Severity: high

   Probability: high

   Possible solution: Source files can be pre-parsed and held in more convenient format in order to save time.

3. ***Appropriate Knowledge Managemet System.***

   Problem: To implement a good KM system takes time and is a useless job, because there are a lot of them on the internet published under GPL licence. The risk is can find an extensible one which would meet our needs?

   Severity: high

   Probability: low

   Possible solution: JSPWiki is considered as the most suitable one.

4. ***Is the wiki system stable and working?***

   Problem: During the course of development, the chosend wiki system might become unusable (for any reason).

   Severity: medium

   Probability: low

   Possible solution: The risk must be anticipated during the design phase. Since our wiki system is designed as plugins, we would probably choose plugin based wiki system.

6.3 Solving System's Risks

Designing is a process with iterations, so first of all the risks with highest severity must be solved.

6.3.1 Knowledge Management System

Open Source community have made a lot of similar projects most of which are unfinished. For example there are dozens of Data Base engines available with almost same features, but you could hardly find one which could compete with Oracle[1] solutions. Our aim was to make a useful solution, not to start a project and leave it unfinished and useless. So we decided to take an existing system and extend it to meet our vision (section 1.4.1).

Since the very beginning of the project a Wiki (what is wiki we described in section 1.5) engine is considered as one of the most suitable types of application to extend. It is because all the project's team like the philosophy of these engines: the way documents are organised and edited in them.

When choosing a concrete Wiki engine to use, these characteristics were considered as most important ones:

Java-based. Only java-based were considered, because we had to comply with our learning subjects.

Extendable. It is much easier to extend a programme when authors has anticipated a possibility to do it.

Project's activity. Since we want to make extentions useful for others, we chose an engine according to its popularity and development activity.

Open Source Project. We were looking exclusively for an Open Source project, because we wanted to sustain an advantage of the user-community.

After some research JSPWiki[2] engine was chosen because it best complies with all the requirements listed above.

[1] http://www.oracle.com
[2] http://www.jspwiki.org

6.3.2 JSPWiki Plugins

There is a problem that it is impossible to anticipate and implement all the features customers may need in a programme you create, so extention points are required. On the other hand when you create a bigger system, you face a need to structure it. To solve these kind of problems a *Plugin* design pattern has been developed and is widely used now.

The main idea of this design pattern is to reduce functionality of the main system. The core part sometimes can even be limited to only maintaining plugins and communication between them. Therefore the rest of the functionality and any extentions are developed as plugins.

The main advantages of a plugin-based system from the implementation point of view:

- ***Independent system.*** It is easy to rewrite just part of a system without worrying about other ones.
- ***Extendibility.*** Usually you can add extra feature without rewriting any code.

The main drawback of this design pattern:

- ***Limited extendibility.*** You are limited functionality provided by the core system, which can be not enough in some situations. For example it would be difficult to implement software a function like *licencing model* (i.e. limit software usage depending on a sertain licence custumer has bought) as a plugin.

JSPWiki is designed in this way too. It is able to write plugins which can be executed from anywhere in a document with some parameters. The output of the plugin can be only a pure HTML[3] code which will be inserted instead of the plugin call.

Since JSPWiki poses good enough plugin system, the Source Documenting part of our project can be implemented as plugins.

[3]Hyper Text Markup Language – a standard for creating internet pages.

6.3.3 ANTLR[4] – ANother Tool for Language Recognition

Since one of the biggest part of the project is *Source Documentation* system, parsing of a programming source which is being documented is unavoidable. For example if you want to provide a inheritance diagram for a particular class, you must to find its parent class. It can be done only by parsing source code.

Source Parsing – is an action of analysing source code of a programming language, extracting needed information and providing it in a more convenient way for further usage.

The complexity of source parsing depends on a programming language. For example to write a parser for *Java* is very easy comparing to $C++$, to write parsing routines for which can take even few months according to sources on the internet. On the other hand, there are already written good tools which are shared under *GPL* (licencing models are described on chapter 3.4.3) or other free licence on the internet.

ANTLR is highly ranked by developers (e.g. SUN[5]) because of its flexibility. It is a tool which can be and is used for writing source parsers for almost any programming language. So once integrated in any system it gives huge value. Though flexibility costs too. Such parsers provide very universal data structures which must be configured in order to make them useful.

ANTLR suits project's needs, because it helps to fulfil Non Functional requirement 2 to document code for as many programming languages as possible. This system also solves the risk 1 about Source Parsing Complexity described in chapter 6.2.

6.3.4 Parsing Source Code in Advance

On a heavy-load systems as less job must be done in order to render a www page as possible. As mentioned before, source code must be analysed in order to make a diagram, which takes too much time to do it every time when source structure is needed.

A solution could be to store parsed source structure in any inner format and read from it when needed. Java has got a feature to serialise and deserialise

[4]http://www.antlr.org
[5]http://www.sun.com

objects so they can be restored fast enough. In this case parsing could be done at a scheduled time, e.g. every night. Though this solution brings another problem of data inconsistence. For example if source code is altered during the day, then diagrams do not reflect current situation and can mislead a user. There are two ways to fix it:

1. Ask user to run source parsing subsystem manualy when he/she alters code.

2. Whenever a structure of a source file is needed (when diagrams are being drawn) system can check whether any changes has been made (it can be done simply by comparing file's modification time) to it and reparse it if so.

The second solution will be chosen, because it better ensures data consistence and makes programme more userfriendly since it reduces tasks needed to maintain the system.

6.3.5 Data Caching

Continuing optimising the system, caching can be introduced. Since these strategies can support each other both should be used in the system. Reading from a hard disk and writing to it is very slow comparing to using system memory. So if the amount of reading and writing operations to disk would be reduced it could lead to a significant performance boost.

Caching Source Structures

In last chapter it was decided to store source code structure in an inner format. Since demand of particular files can be high, it would be wise to cache them – hold a copy in system memory and reuse instead of loading from disk.

System memory is small so everyting would not fit in it. Therefore cache must be organised and useless stuff must be removed from it. Theoretically the best of all it would be to remove object which you will not need any more. The problem is that it is impossible anticipate the future, so it is impossible to know which objects will not be need any more as well. In practice usually objects

which have not been used for the longest time are considered as unuseful ones and are removed.

Caching Diagrams

People usually visit some sites more often then others. So if a site contains some diagrams and is visited quite often it would reduce a system response time and server load if diagrams could be reused.

This leads to an other caching strategy. A list of already generated diagrams could be held and checked before producing a new diagram. Using this improvement blindly can cause data inconsistency problems. It must be kept in mind that source can be altered at any time and checked whenever diagram is being reused.

If implemented propertly this strategy can give a bigger benefit then Source Structure Caching. Since these strategies can support each other both should be used in the system.

6.4 System Architecture

When system's requirements are clear and solutions for all the risks are found, system can be designed. During design phase system must be specified and documented so programmers could take the flour. Usually system is analysed using *Top-Down* paradigm. This method is convenient because you don't have to design all the system at once, so you will not have lost among many classes.

6.4.1 System Layout

First of all programme is divided into more or less independed parts – *components* – and interfaces must be specified for communication among them. The level of independability can be measured according to how easily they can be replaced with other similar ones. For example *Data Base* engines or *Web severs* usually feature similar and strictly defined interface so can be changed quite easily.

In the figure 6.2 you can see, how this system is divided into components.

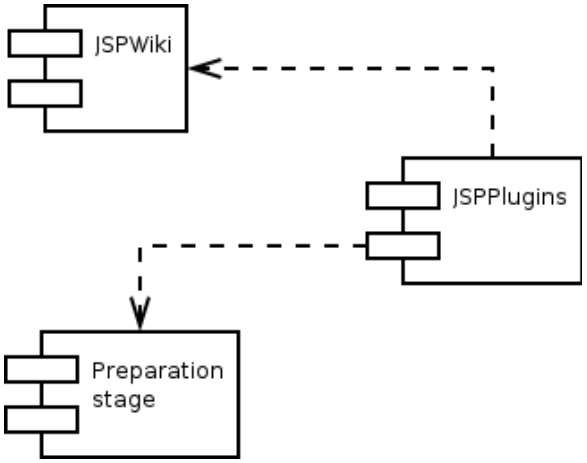


Figure 6.2: System Layout Diagram

JSPWiki- the programme which has been taken and extended during this project.

WikiPlugins- the extentions which has been developed during this project.

Preparation Stage- tools for preparsing source code (see section 6.3.4 for more information).

6.4.2 Class Diagram

When analysing a system for a first time, class diagrams can be very useful. They contain information about class relations in the system and you can get a good picture of it. When there are a lot of classes these diagrams looks scary at the first glance. For the simplicity reasons derived classes can be skipped when drawing a class diagram.

There is a class diagram of our system in figure 6.3. It is stripped down a little bit by skipping *Exception* classes. There are two entry points (external classes) to the system, they are decorated in red in the diagram:

Update – it must be executed when system is deployed or reconfigured.

PlugIn – it is just an example class how any plugin should go into system. This class is executed by *JSPWiki* when a diagram has to be drawn.

Description of other more important classes:

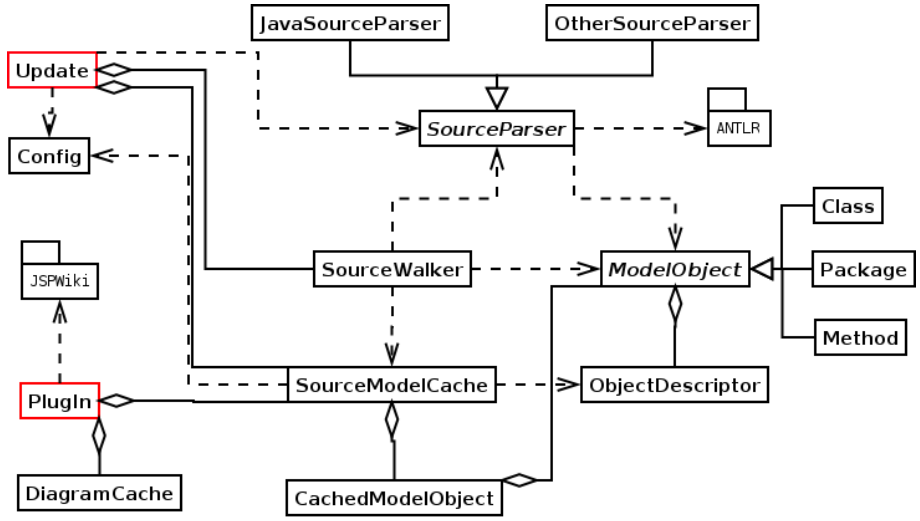


Figure 6.3: Class diagram

SourceWalker – an engine class for walking through a source repository.

SourceParser – it is an abstract class and all class, which are responsible for source parsing, should be derived from it (e.g. *JavaSourceParser*).

ModelObject – classes derived form this one are used to represent source structure in an internal format. This format is used by the rest of system.

SourceModelCache, DiagramCache – these classes are introduced for the performance reasons. They cache source structure and drawn diagrams respectively.

6.4.3 Drawing a Diagram

Using sequence diagrams it is easy to visualise, what exactly is going on in a system at more difficult situations. In these diagrams it is possible to see how objects interact, what methods are being executed and why.

One of the most difficult and important function of the system is diagram drawing, since most of the risks were identified in it (various possible robustness and data inconsistence problems).

The sequence diagram of this part is shown in figure 6.4. It can be seen that first of all the needed data to draw a picture is being gathered. During the

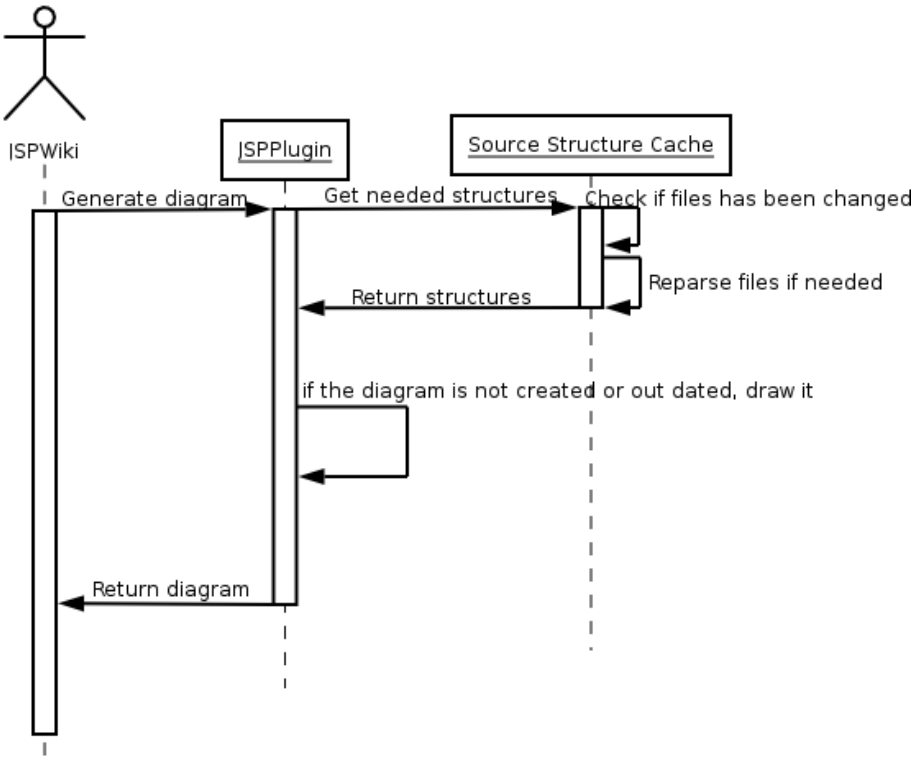


Figure 6.4: Sequence diagram of a Diagram Drawing plugin

data collecting processes it is being checked if files have been changed comparing to prepared ones and being reloaded if needed. Further if there is a diagram drawn a bit earlier and nothing has been changed it is being reused otherwise a diagram is being painted.

6.4.4 Source Parsing

Source parsing is a complex job even when reusing parsers (we use ANTLR parsing package). After parsing a source structure is represented as tree structure. Since a tree presented by ANTLR does not have any more complex methods, they has to be implemente by a user (us).

Few most interesting methods will be described in following sections.

Search

When parsing a tree functions for searching for a node in a tree and a subtree was needed. We impelented a recursive function which finds a subtree first of all if needed, then recursively searches for a node with a specific identification number (i.e. *findElement*).

```
private Vector<AST> findNextElement
    (AST root, Vector<AST> start, int from, int findElement)
{
    AST workingUnit;
    // looking for a subtree if needed (start != null)
    if ((start != null) && (start.size() > from))
        workingUnit = start.get(from);
    else
        workingUnit = root.getFirstChild();
    while (workingUnit != null) {
        Vector<AST> result;
        if ((workingUnit.getType() == findElement) &&
            ((start == null) || (start.size() <= from) ||
            (start.get(from) != workingUnit)))
        {
            // Needed element was found
            result = new Vector<AST>();
            result.add(workingUnit);
            return result;
        }
        result = findNextElement(workingUnit, start, from + 1,
            findElement);
        if (result != null)
        {
            result.insertElementAt(workingUnit, 0);
            return result;
        }
        workingUnit = workingUnit.getNextSibling();
```

```
    }
    // An element was not found
    return null;
}
```

Description of parameters:

root – a tree you want to search in

start – you want to search in a subtree, this list should define a path to it

from – inner variable and should be initialised to 0 at the beginning

findElement – an identification number of a node you a looking for

A path to the first instance of a searching element is be returned as a result if it exists, *null* is be return otherwise.

Concatenating a Subtree

Source code is divided in very small pieces when parsing. For example a package name is split by dots. For example if you have a package *jspdoc.model*, then it will be represented as two nodes in the tree: *jspdoc* with a subnode *model*. It is not convenient since only a full package name interests us.

So there is a need for a method which would concat a subtree. It is a function which recursively concats a given subtree (*subTree*).

```
protected String concat(AST subTree)
{
    if (subTree.getNumberOfChildren() == 2)
        return concat(subTree.getFirstChild()) +
            subTree.getText() +
            concat(subTree.getFirstChild().getNextSibling());
    else
    {
        String result = subTree.getText();
        AST node = subTree.getFirstChild();
        while (node != null)
```

```
        {
            result += concat(node);
            node = node.getNextSibling();
        }
        return result;
    }
}
```

The algorithm differs a bit depending on a situation. For example if a node has got 2 subnodes, then the node should be put in between the subnodes in the result. Otherwise subnodes must be simply concatenated and the node has to be put at the beginning of the result string.

6.5 User Manual

The purpose of this chapter is to get a user acquainted with features this system posses and provide a decent tutorial on them.

6.5.1 System Deployment

Since our system is an extention for an other system, it requires such components to be installed:

Java 1.5 – Java Runtime Environment 1.5 or newer

Apache Tomcat 5.0

JSPWiki – a Wiki engine we have extended

ANTLR – source parsing tool

jspdoc.jar – our system – must be put in $\{JSPWiki\}/WEB-INF/lib/$ directory, where $\{JSPWiki\}$ is a root path of your JSPWiki system.

6.5.2 Configuring

All the configuration of our plugin is kept a single file: *Config.java*. In it you have to set:

sourcePath – a path to your source repository

sourceParser – a parsing class for a programming language of your source. We have made only an engine for Java source

cachePath – a writeable directory, where internal system data could be held

Since *Config.java* is a part of *jspdoc.jar* package, *jspdoc* must be recompiled after configuring.

6.5.3 Drawing an Inheritace Diagram

Anywhere in a JSPWiki document an inheritence diagram of any existing class can be drawn. In order to do that, such line should be inserted (*className* has to be replaced with a name of your class):

```
[{InheritanceDiagram class=className}]
```

For exmaple, for class *jspdoc.source.parsers.java.JavaSourceParser* such a diagram would be drawn:

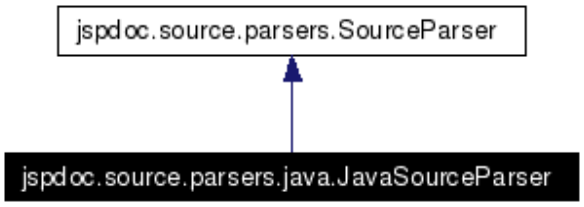


Figure 6.5: inheritace diagram: JavaSourceParser

Good luck using our system !

Chapter 7

The Sequel of our System

Figure 7.1: Gifts of the Great Spirit

So far our system has been analysed from licensing, open source development, business models perspectives. This gives a decent background for further work, but not for immediate actions.

This chapter, herethough, is devoted to the material, which did not have as much attention as others. This is because of time frame given for our project.

As part of this project, we have also glanced a variety of quality model

frameworks, touched Customer Relationship Management and looked at open source project organisation. This could be a great investigative area for future.

7.1 Customer Relationship Management Systems

As we use Free Software, there appears a big demand of relations between users and our company. For this reason we are thinking of CRM as our additional system, which will help fulfil customers' needs.

CRM is a process or methodology used to learn more about customers needs and behaviors in order to develop stronger relationships with them. There are many technological components to CRM, but thinking about CRM in primarily technological terms is a mistake. The more useful way to think about CRM is as a process that will help bring together lots of pieces of information about customers, sales, marketing effectiveness, responsiveness and market trends.

CRM helps businesses use technology and human resources to gain insight into the behavior of customers and the value of those customers.

Customers can be classified into two main groups – internal and external customers. An internal customer is someone who works for the organization, possibly in another department or another branch. External customers are essentially the general public. These two groups can be further broken down:

Internal Customers

- People working in different departments
- People working in different branches

External Customers

- Individuals of different needs or different cultures
- Business people
- Groups

7.1.1 CRM in Business Model

CRM systems influence mainly three business model areas: Resources, Offering and Activities and organization. We will discuss these three levels in the following sections.

Resource Level

A CRM systems is a *resource* that requires both physical and human resources to be implemented and used. The system consists of components that affect other resources of a business model, such as financial, physical, human and organizational resources. A simple version of a CRM system is a Microsoft Outlook. It has the most central and rudimentary functionality of any CRM system: an address book. An address book containing contacts information is the single most important functionality of any CRM system.

Activities and Organization

CRM systems directly affect primary activities such as marketing, sales and service and how these activities are structured and organized. Activities represent the interaction between a business model and its customers and market.

Offering Level

The offering (products or services) and the perception of the offering are affected by adopting a CRM system. It will affect price, cost and profit. For example: some companies sales channel, their Web sites, has configuration system that allows customers to choose design, colors; Our system could be differentiated by employing a skinning mechanism - a method to change software appearnace.

Advantages Of CRM

Using CRM, a business can:

- Provide better customer service
- Increase customer revenues
- Discover new customers

- Help sales staff make deals faster
- Make call centers more efficient
- Simplify marketing and sales processes

These advantages, as of our project nature, are very important.

In next part we discuss the components of effective CRM systems. CRM systems have evolved from the clear business recognition that the customer, and hence management of customer relationships, should be a fundamental element in any corporate strategy. Industry experts loosely call this strategy the customer relationship strategy (CRS).

7.2 Customer Relationship Strategy

CRS consists of two components: traditional CRM systems aimed at maximizing interaction with the customer; and Customer Intelligence (CI) systems aimed at analyzing information to gain knowledge of customer attributes. The technology marketplace has helped blur the lines between these components.

The primary goal of the CRS approach is to maximize corporate strategic goals by gaining in-depth understanding of the organization's position in the market. According to today's best practices, it is imperative that companies embrace the principles of the CRS paradigm at all levels of the organization, from senior management to the line worker. Only then will the company move to a consumer-centric business model that can effectively analyze information and strategically act to achieve its goals.

It is necessary to understand the difference between the goals of these two systems in order to employ successful CRS approach.

Goals of CRM Systems

Traditional CRM systems developed as an evolutionary process originating from corporate customer support systems. These systems help manage customer service by placing customer demographic information in corporate databases, which in turn are utilized by applications such as:

- Sales force automation.

- Marketing and campaign management.
- Supply chain management.
- Call or service center operation.
- Contact or customer prospecting.

Goals of CI Systems

On the other hand, CI systems evolved from the need to analyze customer information to gain expanded insight into the enterprise marketplace. Traditional business intelligence (BI) systems led the way by creating tools to work on data collected from traditional databases as well as non-traditional systems such as e-mail, personal contact histories, and office documents. CI systems were and advancment over BI systems.

CI systems add the crucial concept of information analysis. These tools process information with respect to the environment from which it was gathered. By continuously analysing data gained from customer contact points or touches, CI systems provide insight into customer behaviors, preferences, operations, loyalty, assets, and more. In essence, they aim to answer what are parameters of the sales process.

Any CI system should address these key attributes:

- The need to clearly identify information of value.
- The identification of the context in which the data was gathered or processed. For instance, an increase in umbrella sales may be due to an increase in local precipitation rather than a fashion trend.
- The need to clearly distinguish and associate between different data instances. For example, information surrounding the attributes of customer A may not apply to customer B.

Conclusion

We designed our CRM solution around ease of use. That means company have more time to spend with it's prospects and customers and less time messing with

administrative chores. While the process of gathering customer intelligence has not reached maturity, enterprises should think twice before bypassing CI or giving up on it. It is impossible to build an effective customer relationship strategy on the strength of CRM alone. Enterprises must leverage customer data to gain an in-depth understanding of the marketplace and thus maximize corporate strategic goals.

7.3 Quality Model Frameworks

Quality models are important from two perspectives – one is to validate our own quality, the other is to analyse from business perspective, that is "What possible quality frameworks our system might help achieve". The latter aspect is probably very dependent on what market sector we focus – are we targetind small, or perhaps medium or big sized companies.

Imagine a case where a company (which is/or consider to be our customer) is producing some products, which must be ISO sertified. This in turn means all their tools quality must also be evaluated.

This poses a problem of how to improve and measure open source project quality. Probably, to evaluate our own teams CMM level higher than initial would not be possible. CMM is based on a collection of best practices for software development and maintenance. On the contrary – open source development style is based on agility and adaptiveness.

Maintenance, on the other hand, is another issue, and quality frameworks come handy here and must be included in our business strategy. Unfortunately, this will be left for our fifth semester.

Next, we briefly describe the various quality frameworks.

7.4 Quality Framework Overview

There is a host of quality frameworks for IT companies. While there is some overlap among quality frameworks, in most cases, they don't conflict. Indeed, most large companies use two or three of them. For example, IBM uses ISO 9000, CMM, ITIL, Six Sigma and several homegrown quality programs. Meanwhile,

other equally sophisticated companies don't use any of them, preferring to roll their own. For instance, MasterCard International Inc. has adapted parts of a number of programs to its own way of doing business. It underwent an external assessment for CMM a year ago and implemented some ideas from that, but it hasn't adopted the framework formally. For some companies, an outside body's stamp of approval, such as an ISO 9000 or CMM certification, may be an important factor.

IT Infrastructure Library (ITIL)

ITIL is mainly a collection of best practices for IT service management and operations (such as service-desk, incident, change, capacity, service-level and security management). It is well established, mature, detailed and focused on IT production and operational quality issues.

It is not geared for software development processes, and for such can be combined with CMMI to cover all of IT, not only IT services.

ITIL tracks problems in IT service areas such as help desk, applications support, software distribution and customer – contact system support, and it overlaps CMM in certain areas such as configuration management. ITIL tracks the changes made to operational systems, but the quality of those changes – in terms of the number and severity of problems resulting from them is more a CMM metric.

Capability Maturity Model Integration (CMMI)

CMMI is a more comprehensive process-maturity framework that combines SW-CMM with broader disciplines in systems engineering and product development.

The CMMI extends and combines the Capability Maturity Model for Software (SW-CMM), the Systems Engineering Capability Model and the Integrated Product Development Capability Maturity Model. SW-CMM is a collection of best practices for software development and maintenance. It allows companies to assess their practices and compare them to those of other companies. The SW-CMM measures process maturity, which progresses through five levels: Level 1 – initial, 2 – managed, 3 – defined, 4 – predictable and 5 – optimizing.

Control Objectives for Information and Related Technology(CobiT)

CobiT is an audit-oriented set of guidelines for IT processes, practices and controls. Geared to risk reduction, focusing on integrity, reliability and security. It mainly addresses four domains: planning and organization, acquisition and implementation, delivery and support, and monitoring. Has six maturity levels, similar to CMM's.

The good side of CobiT is it can address risks not explicitly addressed by other frameworks and to pass audits. Works very well with other quality frameworks, especially ITIL.

Some people call it "an IT governance tool" to help IT managers understand what controls are needed and how to measure the effectiveness of those controls. CobiT fits in nicely with CMMI, with CobiT pinpointing the need for certain controls and CMMI putting them into place.

Six Sigma

Six Sigma is a statistical process improvement method focusing on quality from a customer's or user's point of view. Defines service levels and measures variances from those levels. Projects go through five phases: define, measure, analyze, improve and control. The Design for Six Sigma variant applies this method's principles to the creation of defect-free products or services, rather than the improvement of existing ones.

Six Sigma's strengths are that it is a data-driven approach to finding the root causes of business problems and solving them and takes into account the cost of quality. In IT, best applied for relatively homogeneous, repeatable activities such as call center or help desk operations. Its weakness is that it was originally designed for manufacturing environments and it is difficult to apply where process are not already up and well-running. It can improve a process but doesn't tell you if you have the right process to begin with.

ISO 9000

ISO 9000 is a set of high-level, customer-oriented, auditable standards (ISO 9000, 9001 and 9004) for quality management systems. Intended to ensure control, repeatability and good documentation of processes (not products). It

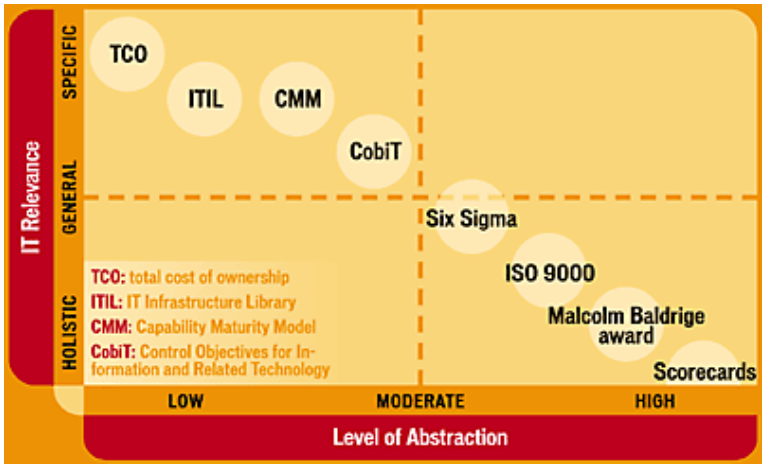


Figure 7.2: Quality Framework Summary

is a well-established and mature framework. It focuses on repeatability and consistency of processes, not directly on the quality of those processes. Not good for analyzing a process and finding root causes of problems.

Malcolm Baldrige National Quality Program

It is a high-level framework for quality in seven areas: company leadership, strategic planning, customer and market focus, information and analysis, human resources, process management and business results. Rates each of these, in terms of approach, execution and results, on a scale from 0 to 100.

The approach is very broad, holistic scope. Can be used by any organization and sit on top of other, more focused IT quality programs.

7.5 Quality Framework summary

All of the before discussed quality frameworks can be summarized in a single chart. The chart approximatelly evaluates framework's relevance to IT area and the level of abstraction.

As for our project, we are interesting in two areas – service quality and software quality.

It is doubtfull if we can use CMM for software quality management, because of it's incompatibility with agile approaches. But probably IT infrastructure

library would support our delivered service quality.

QualityModelMania[1]

7.6 Open Source Organization

Open source project organization is important from the two perspectives – how are going to set-up a project and make it succesful; the second greatly relates to quality frameworks – if we are to endevour on reaching some level in any of previously described quality frameworks, than to understand and deliberately organise is a big importance.

At http://greg.abstrakt.ch/docs/OSP_framework.pdfe found a comprehensive study on open source projects. Therefore, we present here a brief summary of related topics for our system.

7.6.1 Open Source Stages

Open source projects have a common terminology on defining projects maturity. If the businesses wants to use our product, this comes very important.

The various project stages are as follows:

1. Planning

   No code has been written, the scope of the project is still in flux. The project is but an idea. As soon as tangible results in the form of source code appear, the project enters the next stage.

2. Pre-Alpha

   Very preliminary source code has been released. The code is not expected to compile, or even run. Outside observers may have a hard time to figure out the meaning of the source code. As soon as a coherent intent is visible in the code that indicates the eventual direction, the project enters the next stage.

3. Alpha

[1] www.computerworld.com/developmenttopics/development/story/0,10801,90797,00.html

The released code works at least some of the time, and begins to take shape. Preliminary development notes may show up. Active work to expand the feature set of the application continues. As the amount of new features slows down, the project enters the next stage.

4. Beta

   The code is feature-complete, but retains faults. These are gradually weeded out, leading to software that is even more reliable. If the number of faults is deemed low enough, the project releases a stable version, and enters the next stage.

5. Stable

   The software is useful and reliable enough for daily use. Changes are applied very carefully, and the intent of changes is to increase stability, not new functionality. If no significant changes happen over a long time, and only minor issues remain, the project enters the next stage.

6. Mature

   There is little or no new development occurring, as the software fulfills its purpose very reliably. Changes are applied with extreme caution, if at all. A project may remain in this final stage for many years before it slowly fades into the background because it has become obsolete, or replaced by better software.

In evaluation of various wiki engines, JSPWiki was one of the project considered in Stable stage. This is a great adavantage. On the contrary, our own system (a collection plugins) is still between Planning and Pre-Alpha stages.

7.6.2 OS Project Resources

Although Open Source projects differ from classical projects in their resource needs, they still do require them, despite overblown accounts of “virtual organisations”.

Hence, here we briefly outline the resources which must be taken into consideration when establishing real project.

Software resources

Few open source software start from scratch (unless the project is small). Teams also need tools and development platforms to work on.

Our own project is dependent on various software. This includes the free operaing system Debian Linux, Sub-version source countrol system, Dia – free diagram drawing tool, LateX – free superior typesetting tool, MikTex – a free editor in connection with Latex, Eclipse – Java integrated developmet environment, Apache Tomcat – web server and some free web-browsers.

To set-up a real project, we would require much more. Bug-reporting tool, team management tool, and others.

Hardware

The most important resource in the project is Internet. Without internet connection, no access to the world, no access to user and customer community. A capable web server hosting a project web-page, project code-repository and other tools would be required.

People and Coordination

Without people open source project is not an open source project. It can be said open source project is poeple oriented.

Poor coordination is one of the main factors inhibiting the growth of a successful OSP. Only very few individuals in any given project are knowledgeable enough about the finer points of a project to assist with the coordination of work. These persons are usually overwhelmed with work.

Chapter 8

Final Word

The projects report strength is its vast covered area. Indeed, we have glanced at many of the topcis ranging from software licenses, free software movement, system development and business models. The joined view of all these issues requires a great deal of abstract thinking.

The possible worst thing we could have done is to say the non-truth. Hmm... Hmm... So, em, if we did, please excuse as and please give a note to us. Write an e-mail to vilmuciukas13 at gmail.com. If you disagree, or feel we made some minor or major mistakes, we will not argue we did not make them. If we did – than we did, and please let us know. Vi ses!