TILBURG UNIVERSITY

Finding Open options

An Open Source software evaluation model with a case study on Course Management Systems

Master Thesis

Karin van den Berg

open source™

Tilburg, August 2005

Finding Open options

An Open Source software evaluation model with a case study on Course Management Systems

Master Thesis

Karin van den Berg
596809
thesis@karinvandenberg.nl

August, 2005

Supervisors:
drs. Leo N.M. Remijn
dr. ir. Jeroen J.A.C. Hoppenbrouwers

This Master thesis was written as part of the Information Management and Science program at Tilburg University. The thesis defence will take place on the 23rd of August, 2005 in room B834 at Tilburg University.

Summary

The Open Source software market is getting more and more attention. Large IT corporations such as IBM and Novell are investing in Open Source software. Open Source software development is very different from traditional proprietary software. In order to understand Open Source software better, this thesis offers a model for Open Source software evaluation, which can be used as a tool to find the right software package to meet the user's needs.

This research project was performed at Tilburg University in the Department of Information Systems and Management. The goal was to get a better understanding of Open Source software and to make the Open Source software process more understandable for those who evaluate this type of software.

An introduction to Open Source software is followed by the Open Source software evaluation model, using the criteria found in Open Source literature.

Community – the driving force behind an Open Source project
Release Activity – showing the progress made by the developers
Longevity – how long the product has been around
License – is one of the general Open Source licenses used
Support – from the community as well as paid support options
Documentation – user manuals and tutorials, developer documentation
Security – responding to vulnerabilities
Functionality – testing against functional requirements
Integration – standards, modularity and collaboration with other products
Goal and Origin – why was the project started and what is the current goal

These criteria form the key terms of the model. The evaluation process is described using these criteria. The practical part of the model consists of two steps. In the first step selection on the candidate list is performed, using four of the above criteria: Functionality, Community, Release Activity and Longevity. These criteria were selected because they can be evaluated quickly for each candidate in order to eliminate non-viable candidates and select the best ones. This step results in a 'short list' of candidates that can be evaluated in depth in the second step, taking a closer look at the software and the project using all ten criteria.

In order to test this model on real Open Source software, a case study was performed on Course Management Systems. In this case study the model is applied on a candidate list of 36 systems, and evaluation is performed on the top two systems found in the selection step. This evaluation led to a clear conclusion. The best system in this evaluation is the Course Management System called Moodle. The results of the case study are consistent with real life performance of the Course Management Systems.

Nederlandse Samenvatting

De Open Source software markt krijgt steeds meer aandacht. Grote IT bedrijven zoals IBM en Novell investeren in Open Source software. Open Source software ontwikkeling wijkt sterk af van de traditionele 'proprietary' software. Om Open Source software beter te begrijpen biedt deze scriptie een model voor Open Source software evaluatie, wat gebruikt kan worden om de juiste software te vinden.

Dit onderzoeksproject is uitgevoerd aan de Universiteit van Tilburg op het departement Informatiekunde. Het doel was om een beter inzicht te krijgen in Open Source software en deze begrijpelijker te maken voor het evalueren van deze software.

Een introductie van Open Source software wordt gevolgd door het Open Source software evaluatie model, gebruik makend van de volgende criteria uit de Open Source literatuur.

- Community – de drijvende kracht achter een Open Source project
- Release Activiteit – laat de voortgang van de ontwikkelaars zien
- Levensduur – hoe lang bestaat een product al
- Licentie – gebruikt het product een van de publieke Open Source licenties
- Support – van de community en betaalde opties
- Documentatie – handleidingen voor gebruikers en ontwikkelaars
- Veiligheid – reageren op veiligheidsgaten
- Functionaliteit – het testen van de vereiste functionaliteit
- Integratie – Standaarden, modulariteit en samenwerking met andere producten
- Doel en oorsprong – waarom is het project gestart en wat is het huidige doel

Deze criteria vormen de kernbegrippen van het model waarmee het evaluatieproces wordt doorlopen. Het praktische deel van het model bestaat uit twee stappen. In de eerste stap wordt selectie toegepast op de lijst kandidaten, gebruik makend van vier van de bovengenoemde criteria: Functionaliteit, Community, Release Activiteit en Levensduur. Deze criteria zijn geselecteerd omdat ze snel kunnen worden geevalueerd voor elke kandidaat om de ongeschikte kandidaten te elimineren en de beste systemen te selecteren. Uit deze stap komt een 'short list' van kandidaten die verder kunnen worden geëvalueerd in de tweede stap, waarbij de software en het project nader worden bekeken met behulp van alle criteria.

Om het model te testen is een case study uitgevoerd op Course Management Systemen[1]. De selectie is toegepast op 36 kandidaten. De twee beste systemen uit de selectie zijn geëvalueerd. Deze evaluatie leidde tot een duidelijke conclusie waarbij het Course Management Systeem genaamd Moodle het beste systeem is. De resultaten van de case study kwamen overeen met de werkelijke prestaties van de systemen.

[1] ook wel bekend als Digitale Leeromgeving (DLO)

Preface

When I approached my third year thesis supervisor drs. Leo Remijn about conducting a research project for my Master Thesis in the summer of 2004, I knew I wanted a subject in the line of software programming. I have been interested in programming for some time now, something that evolved while following several subjects followed here at Tilburg University. We agreed on a subject to do with programming in combination with the subject of E-Learning. His suggestion: Open Source Course Management Systems. Upon investigating this subject I found some interesting projects. Eventually my interested shifted in the direction of Open Source software in general. This very interesting but not much chartered subject in the academic world grabbed my interest with force. A new idea was born, which resulted in the thesis you are now reading.

I would like to thank first of all my supervisor, drs. Leo Remijn, for his support and advice during my time at the department. He has been very supportive of my interests and has given me great supervision and advice during all those months.

I would also like to extend my gratitude to the people at the department 'lab', the place I spend four days a week for the past nine months. I had a great time working there and they have been very helpful, giving advice on a large variety of subjects from study related things to life itself. In particular I would like to thank Jeroen Hoppenbrouwers for helping me in getting my ideas to take shape and giving lots of advice, and Kees Leune who has been very helpful in practical things as well as giving advice and sharing knowledge with me and my lab coworkers. My lab coworkers also deserve a note of thanks for the time we have had during my thesis work at B737. Sebas, Maarten, Ivo and Jeroen: Thanks for the great times!

Finally I would like to thank my mother and father who have always been very supportive of my goals, and last but certainly not least, my boyfriend of eight years, Werner, who's patience and support during the six years of my university work has been tremendous.

I have had a wonderful time doing the research and writing this thesis, and I have learned a lot that will be very valuable in my next step as a freelance programmer.

Karin van den Berg
Tilburg, August 15 2005

Contents

Summary ii

Nederlandse Samenvatting iii

Preface iv

1 Introduction 1
- 1.1 Objective and Research Question . . . 1
  - 1.1.1 Scope . . . 2
- 1.2 Method . . . 3
  - 1.2.1 Literature . . . 3
  - 1.2.2 Model . . . 3
  - 1.2.3 Case Study . . . 3
- 1.3 Outline . . . 4

2 About Open Source Software 5
- 2.1 Source Code . . . 6
- 2.2 Open Source software development . . . 6
- 2.3 Free or Open Source? . . . 7
- 2.4 Proprietary Software . . . 7
- 2.5 Culture & Movements . . . 7
- 2.6 Unique characteristics . . . 9

3 Open Source Software Evaluation 10
- 3.1 The Criteria . . . 10
  - 3.1.1 Goal and Origin . . . 11
- 3.2 Description of the criteria . . . 12
  - 3.2.1 Community . . . 12
  - 3.2.2 Release activity . . . 12
  - 3.2.3 Longevity . . . 13
  - 3.2.4 License . . . 14
  - 3.2.5 Support . . . 14
  - 3.2.6 Documentation . . . 15
  - 3.2.7 Security . . . 16
  - 3.2.8 Functionality . . . 16
  - 3.2.9 Integration . . . 16

3.2.10 Goal and origin . . . 18
3.3 Selection . . . 18
3.3.1 The Selection Method . . . 19
3.3.2 The Criteria . . . 19
3.4 Evaluation . . . 21
3.4.1 Community . . . 22
3.4.2 Release Activity . . . 22
3.4.3 Longevity . . . 23
3.4.4 License . . . 23
3.4.5 Support . . . 23
3.4.6 Documentation . . . 24
3.4.7 Security . . . 25
3.4.8 Functionality . . . 26
3.4.9 Integration . . . 26
3.4.10 Goal and Origin . . . 27
3.5 Model overview . . . 27

4 Case Study: Course Management Systems 29
4.1 Introduction . . . 29
4.2 Selection . . . 30
4.3 Evaluation . . . 34
4.4 Case Study: Conclusion . . . 38

5 Conclusion & Further Research Recommendations 40
5.1 Research Results . . . 40
5.2 Contribution . . . 42
5.3 Recommendations for Further Research . . . 43

Bibliography 44

A The Open Source Definition 50

B Community Activity 53

C Case Study – Functional Requirements 56

D Use of CMSs in Dutch Universities 57

E Candidate List 58

F Selection Results 60

G Moodle Philosophy 69

H ATutor Philosophy 72

I CMS Evaluation 76

J Remarks on validity of case study results 91

Chapter 1

Introduction

'IBM tests new ways to support open source' (Sharma, 2005)
'Indian president calls for open source in defense' (Sharma, 2004)
'Nokia And Apple Collaborate On Open Source Browser' (Carless, 2005)
'California considers open-source shift' (Becker, 2004)

These are just a few of the many news headlines that show the rise of Open Source usage in business and government. Open Source has been getting much attention in the last few years. Many corporations, large and small, have taken an interest in this growing software market that shows some strong differences with traditional software. Open Source is no longer seen as an insignificant niche market but as a serious development in the software market.

Businesses as well as educational institutions can benefit from Open Source software. However, it is still a mysterious and sometimes even scary world for many people. News stories about the risks of Open Source software, whether they are genuine or not, can cause uncertainty with the IT managers that have to make the decisions about the company's IT policy and the software they use.

The Open Source software market offers some great opportunities, but needs to be understood before evaluating the software for use in businesses and institutions. This thesis was written to give those interested in using Open Source software an idea of what to look for in an Open Source project.

1.1 Objective and Research Question

In the research project that was conducted at the Department of Information Systems and Management at Tilburg University, was done to get a better understanding of the Open Source world and the unique approach of Open Source software development. This knowledge can be used in the field of software evaluation.

The goal of this thesis is to give a method of Open Source software evaluation that anyone could use. Most evaluation reports give a summary of the results but say little or nothing about the method used to get those results. With the rapid development of software, especially Open Source software, those results are not valid very long, because the software changes so fast, so the results themselves are not as valuable as the method for getting those results. In this thesis, a detailed description of the

method is given as well as a case study in which a full description is included of how the model is applied.

The research question addressed in this thesis is the following:

'Is it possible to define a software evaluation model specifically aimed at Open Source software, and can this model be applied to select a Course Management System?'

To answer this question, the following subquestions are used:

- Are there characteristics that make Open Source software unique that are relevant to software evaluation, and if so, what are they?
- Which criteria can be defined for Open Source software selection and evaluation?
- What information is needed to score these criteria?

In order to answer these questions the Open Source market will be studied using literature and information from the Internet. With this knowledge a model for Open Source software evaluation is created that will give insights into the unique characteristics of Open Source software relevant to those who intend to use it. This model should give an answer to the question which Open Source software project in a software category is best suited for use in the evaluator's business or institution.

To test the constructed model in a real life situation a case study will be performed in which this model is used. The target software category is Course Management Systems. The central question for the case study is:

- How well does this model apply to evaluation of Course Management Systems?

The market of Open Source Course Management Systems (CMSs)[1] is explored using the evaluation model. CMSs are being used more and more in universities to support face-to-face classes. However, aside from the two proprietary market leaders, Blackboard and WebCT, not many other systems are being used as of yet. The Open Source market may offer viable alternatives.

1.1.1 Scope

The systems this model is aimed at can vary from small software packages to large systems. However, for very small software that does not perform a critical function, following the whole model is probably too elaborate, but certain parts may still be useful. The case study tests the model on a large type of system, the Course Management System. This type of system offers a range of functionality for a large number of users that use the system simultaneously.

This thesis is not about comparing Open Source software to proprietary software. It is about the unique characteristics of Open Source software development and how to evaluate this software in terms of these characteristics, what things require attention when evaluating software in the Open Source market.

[1] Not to be confused with *Content* Management Systems, also often abbreviated as CMS

Open Source software is in most cases developed in a very open environment. Information that would not be available in a proprietary setting can be used when evaluating Open Source software to give a better picture of the software and the project that brings it forth (Wheeler, 2005). It also gives a better idea of the potential continuity of the project. Traditional software evaluation methods focus mostly on the software itself, the functionality, usability and price. With the model that is presented here, a much broader approach is used for evaluating Open Source software using the additional available information.

1.2 Method

The method used to construct this thesis consists of three parts: literature study, construction of the evaluation model and the case study.

1.2.1 Literature

Open Source is a relatively new field of study in the academic world. The number of articles written on the subject is not high.

To start the literature study, general Open Source articles and information is used to get a good global idea of Open Source software. This literature will also be the basis for the first chapter. In order to establish the criteria for the evaluation model, literature on comparing Open Source software and other models such as Open Source software maturity models are consulted.

Because scientific literature from the academic libraries may not offer enough to complete the research, other sources need to be explored as well, by using search engines on the Internet, for example. Of course the validity of non-scientific resources is verified.

1.2.2 Model

From the literature mentioned above, the Open Source evaluation criteria are identified. These will be the key terms of the evaluation model for Open Source software.

The model consists of two parts. The first part describes the background of the criteria, what effect they have on the Open Source software project and why they are important. In the second part the evaluation process is described, explaining the method for identifying the key values for each criterion.

1.2.3 Case Study

The case study will be performed on Course Management Systems. This software category was chosen because it concerns reasonably large applications, targeted at institution wide use at educational institutions, which warrants a full evaluation. A list of candidates is constructed using comparison reports and websites on CMSs. The model created in the previous step will be followed to identify the highest scoring systems. The evaluation process and the observations made are described. The final result is calculated for each system and a ranked list is made of all systems. These

results are compared to existing performance results of these systems, in terms of adoption and performance in other comparisons, to determine whether the model leads to an acceptable result.

1.3 Outline

Figure 1.1 gives a graphical representation of the structure of this thesis.

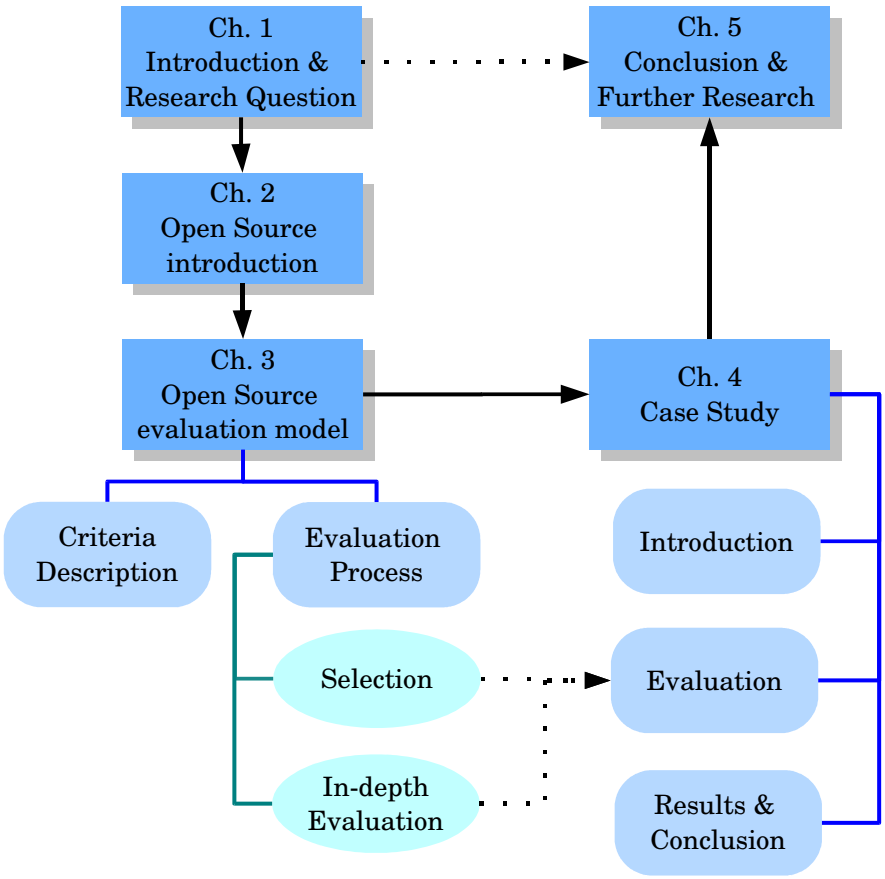

Figure 1.1: Thesis Outline

Chapter 2 is an introduction to Open Source, including the Open Source development method and the Open Source culture. In chapter 3 the evaluation model is discussed, starting by a description of the criteria in section 3.2, followed by a description the selection process in 3.3 and the in-depth evaluation process in 3.4. In chapter 4 the case study that has been performed will be outlined, starting in 4.1 with an introduction on Course Management Systems and the functional requirements set for the CMS in this case study. The model is applied to the candidates, following the selection process to select the top two candidates in 4.2 that are evaluated in-depth in 4.3. The case study is closed by the presentation of results and the case study conclusion in 4.4 where the results are validated. Finally, the Conclusion answers the research question in 5.1, and recommendations for further research are made in 5.3.

Chapter 2

About Open Source Software

Although Open Source software has existed since the 1960's (Weber, 2004, chap. 2), only in the last few years has it gotten much attention. In 1983 the Free Software Foundation was founded by Richard Stallman (Hars and Ou, 2002). The term 'Open Source' was introduced in 1998 (Raymond, 1998a). Since then more and more companies have taken an interest in Open Source software. Recently Novell acquired Suse Linux, one of the distributions of the Linux operating system, taking their embrace of Open Source a step further (Novell, 2003), and with this expanding the enterprise market for Linux.

Linux and Open Source are often linked in Open Source literature, and it may seem that Linux and its added software is all there is in the Open Source market. However, Open Source software is much more than Linux and Linux-compatible software (O'Reilly, 1999). Many more examples of Open Source software exist, such as the Apache web server, with a market share of almost 70% (Netcraft, 2005), the web language PHP, the database server MySQL, the office suite OpenOffice.org and a very large number of web applications (Wheeler, 2004).

The source code, the instructions that make up the 'recipe' for a software package (Weber, 2004, p.4), is freely available to its users in the case of Open Source software. The term Open Source is defined by the Open Source Initiative (OSI) in the Open Source Definition (OSI, 2002). The full definition can be found in Appendix A and can be summarised as:

- The software must be freely distributable
- The source code must be included in the distribution or there is a well-publicised method of obtaining the source code
- Derived works and modifications are allowed
- The license must not be specific to a product, not restrict other software and be technology-neutral

The following sections give some background information on Open Source software, the development process and the culture, which helps to understand the Open Source movement as a whole and the unique characteristics of Open Source software that are used in the evaluation process.

2.1 Source Code

The source code of any software is written in a programming language, for example C, Pascal or web programming languages like PHP. This code is written in textual form. The source code consists of instructions that define how a program behaves. The programmer can add or change instructions to adjust the program's behaviour and add functionality. (Weber, 2004, p.4)

Figure 2.1 shows an example of a very basic piece of source code and its result.

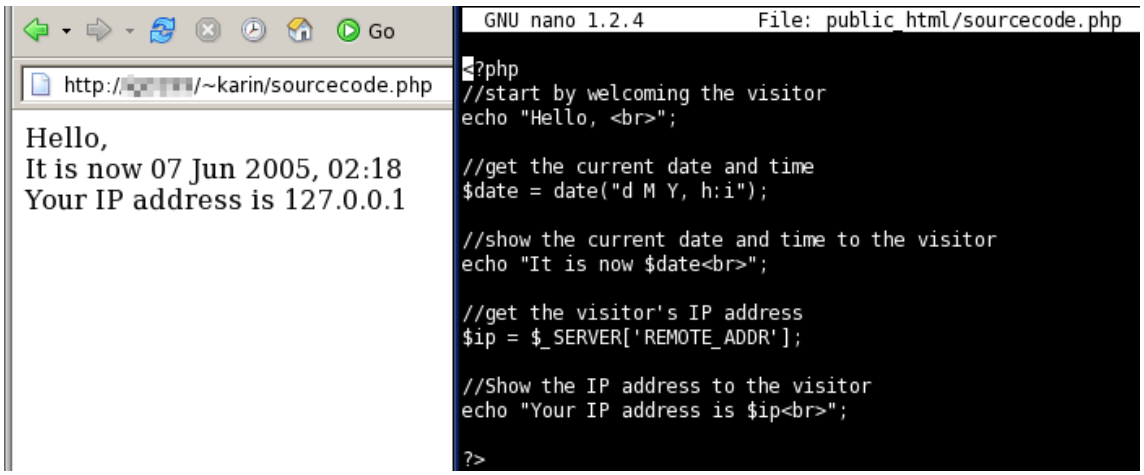

Figure 2.1: Source Code Example

On the right the instructions are shown. This code is written in the web language PHP. By pointing a browser to the file on the server, the result as shown on the left is displayed. The lines in the code preceded by two slashes (//) are comments. These are ignored when the code is executed, and are used to explain what certain code does and why. The words preceded by a dollar sign ($) are variables in which values of executed code can be stored.

2.2 Open Source software development

Open Source software offers the source code along with the software, at no charge[1]. This enables the user to change the instructions of the software, changing its behaviour, adding functionality, and so on. It gives anyone the opportunity to participate in the development of the software project (Wheeler, 2005).

Open Source projects are, in most cases, run on the Internet. In fact, the Internet enabled Open Source projects to form and grow (Weber, 2004, p.83). Open Source projects' websites carry much information: discussions, documentation, bug databases, and so on. This information is very valuable for the evaluation of Open Source software.

Most Open Source projects encourage users to participate in the project in any way

[1]except perhaps distribution costs

they can, from filing bug reports[2] to development of the source code. When the user wants something changed or added to the software, he is at liberty to do it himself, but by working together with the project community and contribute the changes in source code back to the project, the code will be a part of the software for everyone, which will keep it maintained and avoid problems when upgrading the software. If the user keeps the code to himself, he will have to find a way to integrate any updates to the software in with his changes (Glass, 2003). Open Source software developers work together voluntarily to create and improve a product they want to use. They also get a certain satisfaction from being part of the project. There is a number of articles available on the motivation of Open Source software development, such as Hars and Ou (2002) and Hertel (2003).

2.3 Free or Open Source?

The term 'Open Source' was first introduced by Eric. S. Raymond (Raymond, 1998a) in 1998 to eliminate the confusion that came with the term 'Free software' that was being used thus far. Here 'Free' was meant as Freedom, also indicated by the term 'Libre'. Another way to describe it is 'free as in speech, not free as in beer' (Weber, 2004, p.5). However, 'free' is also used for software that is available at no cost, but without the source code being available. This type of software is often labelled as 'freeware'.

To this day there are still advocates of using the term 'Free/Libre software' instead of 'Open Source software', mainly on the side of the Free Software Foundation (FSF) (FSF, 2005b)

2.4 Proprietary Software

Software that is not Open Source is also referred to with different terms, such as commercial software or proprietary software. Some will argue that commercial software can still be Open Source software, since it can be used commercially (FSF, 2005a). 'Proprietary' is a term used in many documents on Open Source, indicating the 'closed source' software. Proprietary Software is software where the source code belongs strictly to the vendor and is not given to the public.

In this thesis the characteristics that distinguish Open Source software from proprietary software are the main focal point. The model in the next chapter is meant for evaluation of different Open Source systems, not a mix of proprietary and Open Source software or comparing proprietary software to Open Source software.

2.5 Culture & Movements

The Open Source movement is as much a culture as it is a development method. The culture is mostly one of sharing and collaboration, often with people from all over the

[2] a report of a problem in the software that the developers can fix with enough information on how to reproduce the problem

world. However, as was briefly addressed before, there is some discussion within the Open Source movement.

The two key groups in the Open Source culture can be described using the comparison by Raymond (1998c) as 'The pragmatists and the purists'.

He describes two degrees of variation: zealotry – whether something, in this case Open Source, is seen as a means to an end or an end in itself – and hostility to commercial software. The purist is a person of great zeal and high hostility to commercial software. A pragmatist is only moderately anti-commercial and sees open source more as a means than an end.

The purist group can be found mostly in the Free Software Foundation (FSF[3]), founded by Richard M. Stallman (Raymond, 1998c). The FSF has its origins in the GNU[4] group, which has also introduced the GNU GPL[5], a strong expression of the purists' view. The GNU GPL will be discussed further in the next chapter.

The FSF views Free or Open Source software as a right, and non-free software as doing harm (Stallman, 1992), while the pragmatists view Open Source software development as a nice method of creating good software. The FSF promotes four kinds of freedom (FSF, 2005a):

1. The freedom to run the program, for any purpose (freedom 0)
2. The freedom to study how the program works and adapt it (freedom 1)
3. The freedom to redistribute copies (freedom 2)
4. The freedom to improve the program and release the improvements (freedom 3)

A key principle that is also reflected in the license that is used and promoted by the FSF, the GNU GPL, is copyleft. A copyleft license uses copyright to protect the licensed source code from becoming incorporated in closed source software. Anything that is licensed under a copyleft license has to stay under that license, including any derivatives (Weber, 2004, p.48).

The FSF freedoms are largely coherent with the Open Source Definition mentioned earlier, though the FSF believes only in copyleft-type licenses, while the OSI definition also approves non-copyleft Open Source licenses.

The pragmatists group has grown much in the Linux movement. Linus Torvalds, though not opposing Richard M. Stallman, publicly endorsed the use of high quality commercial software (Raymond, 1998c) & (Yamagata, 1997). Bruce Perens, who lead the Debian project (Perens, 2005), also shares a pragmatist view. The Debian social contract's basic principle is non-discrimination against any person, group of people, or field of endeavour, including commercial use. 'Do nothing to slow down the widespread distribution and use of open source software' (Weber, 2004, p.86). The FSF exclusion of anything proprietary is opposed to the principle of the Debian social contract to promote growth of Open Source software usage and development.

[3] `http://www.fsf.org/`
[4] `http://www.gnu.org`
[5] General Public License

2.6 Unique characteristics

Open Source software development differs much from proprietary software development. The unique characteristics also lead to a different approach for software evaluation.

First of all, an Open Source project is created because someone has a certain need for software that does not yet exist. To quote Eric S. Raymond (1998b) '*Every good work of software starts by scratching a developer's personal itch*'. There is a strong personal motivation for the developer, and any developer that thereafter joins the project, to work on the software, making it better. This means that they really use the software they create, which is not always the case with developers creating proprietary software. This is also believed to be a reason why Open Source software is often of high quality (Raymond, 1998b).

There are a number of characteristics that define a project's chance of success. When evaluating software in the Open Source market, these unique characteristics have to be taken into account in order to find the software that is mature enough and has a good chance of survival. In some ways the chances of a Open Source project's existence in the future are more predictable than of proprietary software vendors. A company can stop its activities at any time, for a large number of reasons. Open Source software, especially the larger projects, are the work of a many developers and other community members. Because of the openness, anyone can join in or leave. As long as there are still people interested in the project, it can continue. Because of the openness of the software as well as the project, these things are more measurable, giving a better impression of the software and its future.

In the next chapter the Open Source software evaluation model is discussed.

Chapter 3

Open Source Software Evaluation

In this chapter, a model is proposed for evaluating Open Source software using an approach adopted to the unique characteristics of Open Source software. This comparison can be used stand-alone or in conjunction with traditional software evaluation methods.

The criteria that are used were derived from Open Source literature, among which are two Open Source maturity models. These sources are discussed in the next section.

In order to understand their importance and value, the section 3.2 gives a description of each criterion. The evaluation process, consisting of the selection of a small number of candidates (the 'Short list') and the in depth evaluation of the short listed candidates, is described in sections 3.3 and 3.4

3.1 The Criteria

The criteria for the Open Source software evaluation model were established using Open Source literature on the subject of evaluation of software. Because scientific literature is still somewhat scarce on the subject, web searches were needed to find the required resources. The web searches were done to find the most prominent articles on Open Source maturity models, Open Source success factors and Open Source software evaluation. Two Open Source maturity models were found, as well as three articles giving advice on selecting Open Source software and one research article that investigated Open Source success factors.

In order to identify the criteria that give a good general idea of the Open Source software that needs to be evaluated, all criteria were listed and the terms covering the same areas were grouped together. The criteria that were mentioned in some way in at least four out of the six sources were included in this model. The six sources are listed below.

- Capgemini Expert Letter – Open Source Maturity Model (Duijnhouwer and Widdows, 2003)

- Succeeding with Open Source (Golden, 2005) – This book uses the Navica Open Source Maturity Model
- Towards A Portfolio of FLOSS Project Success Measures (Crowston et al., 2004)
- How to evaluate Open Source / Free Software (OSS/FS) Programs (Wheeler, 2005)
- Ten Rules for Evaluating Open Source Software (Donham, 2004)
- Vijf adviezen voor selectie van oss-componenten (Nijdam, 2003)

The following two tables list the eight main criteria and how they were mentioned in each source. A criterion is either listed as mentioned in the article (Y) or listed as mentioned under a different term (i.e. 'Maintenance') or as part of another section (i.e. 'In support').

Table 3.1: Literature on Open Source software evaluation – Criteria I

Criterion	Duijnhouwer et al. 2003	Golden 2005	Crowston et al. 2004	Wheeler 2005	Donham 2004	Nijdam 2003
Community	Y	Y	Team size and activity level	In support	–	Active groups
Release Activity	–	Activity level	Activity level	Maintenance	–	Active groups
Longevity	Age	Y	–	Y	Maturity	Version
License	Y	In risk	–	Y	Y	Y
Support	Y	Y	–	Y	Y	–
Documentation	In ease of deployment	Y	–	In support	Y	–
Security	Y	In risk	–	Y	Y	–
Functionality	Features in time	Y	–	Y	Y	Y
Integration	Y	Y	–	In functionality	In infrastructure	–

These nine criteria are applicable to almost any type of Open Source software and give a good general indication of the software.

3.1.1 Goal and Origin

One additional criterion that is an underlying theme in Open Source literature such as Raymond (1998b), Golden (2005), Weber (2004) and Hars and Ou (2002) has to do with the motivation of the Open Source software developers. The motivation behind a certain project can explain the rationale for intended use and how serious the developers are about the project. Though this criterion does not have the same weight as the main nine criteria, it is included in this model.

3.2 Description of the criteria

The criteria are each described in the following subsections, giving background information and explaining why the criterion is important for software evaluation.

3.2.1 Community

'One of the most important aspects of open source is the community' (Golden, 2005, p.21).

The user community of an open source project consists of the people that use the software and participate in some way. One way of participating is by filing bug reports, which is a report on a problem in the software. This could be something very small, like a typing error in the text, or something large, like a feature that does not work properly. Another way is giving feedback on functionality the user would like to be added. Some users turn into developers over time, and participate in things such as fixing bugs or adding functionality themselves. They cross the line to the developer community, which is often a line made very thin by encouraging participation and making the developer community accessible to anyone who is interested. In some cases the user and developer community interact fully in the same discussion areas.

The community of an Open Source project is very important because it is the community that does most of the testing and provides quality feedback. Instead of using financial resources to put the software through extensive testing and Quality Assurance (QA), like a proprietary vendor will do, the Open Source projects have the community as a resource. The more people are interested in a project, the more likely it is that it will be active and keep going. A large and active community says something about the acceptance of the software. If the software was not good enough to use, there would not be so many people who cared about its development (Duijnhouwer and Widdows, 2003).

3.2.2 Release activity

The activity level of a project consists of the community activity and the development activity. The community was discussed above. The development activity is reflected in two parts

- Their participation in the community
- The development itself – writing or changing the source code.

The latter activity is reflected mostly in the release activity. All software projects release new versions after a period of time. The number of releases per period and their significance, meaning how large the changes are per release (i.e. are there feature additions or just bug fixes in the release), illustrates the progress made by the developers. This gives a good indication of how seriously the developers are working on the software.

The Open Source repositories SourceForge[1] and FreshMeat[2], where projects can share files with the public, provide information on activity that could be useful to evaluate the release activity (Wheeler, 2005).

An Open Source project often has different types of releases:

- Stable releases – the most important type for the end user. This is the version of the software that is deemed suitable for production use[3] with minimal risk of failure.
- Development versions – These can have different forms, such as 'beta', 'daily builds' or CVS (Concurrent Version System) versions, each more up to date with the latest changes. These versions are usually said to be used 'at your own risk' and not meant for production use because there is a higher possibility of errors.

A project which releases new versions of their software usually publishes release notes along with the download that list all the changes made in the software since the previous release. Other than the release notes, the project might also have a roadmap, which usually shows what goals the developers have, how much of these goals are completed, and when the deadline or estimated delivery date is for each goal. Checking how the developers keep up with this roadmap shows something about how well the development team can keep to a schedule.

Though a project might stabilise over time as it is completed, no project should be completely static. It is important that it is maintained and will remain maintained in the future (Wheeler, 2005).

3.2.3 Longevity

The longevity of a product is a measure of how long it has been around. It says something about a project's stability and chance of survival. A project that is just starting it usually still full of bugs (Golden, 2005, p.103). The older a project, the less likely the developers will suddenly stop (Duijnhouwer and Widdows, 2003). But age is not always a guarantee of the chance of survival. First of all, very old software may be stuck on old technologies and methods, from which the only escape is to completely start over. Some software has already successfully gone through such a cycle, which is a good sign in terms of maturity. One thing that needs to be taken into account when products are not very young is whether or not there is still an active community around it.

The age and activity level of project are often related. Young projects often have a higher activity level than older ones, because once a project has stabilised and is satisfactory to most users, the discussions are less frequent and releases are smaller, containing mostly bug and security fixes. This doesn't mean that the activity should ever be slim to none. As mentioned before, no project is ever static (Wheeler, 2005). There's always something that still needs to be done.

[1] `http://www.sourceforge.net`

[2] `http://www.freshmeat.net`

[3] Production use means using the software in a business-critical manner, for example the actual use of an accounting program in a company

3.2.4 License

As mentioned in the previous chapter, the licenses in the Open Source world reflect something of the culture. The most important term in this context is 'opyleft', introduced by Richard Stallman, where copyright is used to ensure free software and their derivative works remain free (Weber, 2004, p.48). In essence a 'copyleft' license obligates anyone who redistributes software under that license in any way or form to also keep the code and any derivative code under the license, thus making any derivatives Open Source as well.

The most well-known example of a 'copyleft' license is the GNU GPL (Weber, 2004, p.48-49). This is also one of the most used licenses. On SourceForge, a large Open Source public repository where over 62,000 projects reside, almost 70%[4] uses the GNU GPL as their license. There are some large and well known products that do not use SourceForge, and some of these have their own license, such as Apache, PHP and Mozilla (OSI, 2005).

Because 'copyleft' in the GNU GPL is very strong, an additional version was made called the LGPL (Library GPL, also known as 'Lesser' GPL) which was less restrictive in its 'copyleft' statements, allowing libraries to be used in other applications without the need to distribute the source code (Weber, 2004, p. 183).

A 'non-copyleft' license that is much heard of is the BSD license. It has been the subject of much controversy and has had different versions because of that. Components that are licensed under the BSD are used in several commercial software applications, among which are Microsoft products and Mac OS X.(Wikipedia, 2005a)

The license of the software in use can have unwanted consequences depending on the goal of the use. If the users plans to alter and redistribute the software in some way but does not want to distribute the source code, a copyleft license is not suitable. In most cases the user will probably just want to use the software, perhaps alter it to the environment somewhat, but not sell it. In that case the license itself should at least be OSI approved and preferably well known. The license should fit with the intended software use.

3.2.5 Support

There are two types of support for a software product.

- Usage support – the answering of questions on the use of the software
- Failure support or maintenance – the solving of problems in the software

Often the two get mixed at some level because users do not always know the right way to use the product. Their support request will start as a problem report and later becomes part of usage support (Golden, 2005, p.124).

The way support is handled is a measure of how seriously the developers work on the software (Duijnhouwer and Widdows, 2003). One way to check this is to see if there

[4] Established using the SourceForge Software Map on April 20, 2005, `http://sourceforge.net/softwaremap/trove_list.php?form_cat=13`

is a separate bug tracker[5] for the software, and how actively it is being used by both the developers and the users. When the developers use it but hardly any users seem to participate, the users may not be pointed in the right direction to report problems.

Aside from community support, larger or more popular projects may have paid support options. The software is free to use, but the user has the option to get professional support for a fee, either on a service agreement basis where a subscription fee is payed for a certain period of time, or a per incident fee for each time the user calls on support. The project leaders themselves may offer something like this, which is the case for the very popular Open Source database server MySQL (MySQL, 2005).

There are companies that offer specialised support for certain Open Source software. This is called third party support. For example, at the Mozilla Support web page, it can be seen that DecisionOne offers paid support for Mozilla's popular web browser FireFox, the e-mail client Thunderbird and the Mozilla Suite (Mozilla, 2005).

The fact that paid support exists for a Open Source product, especially third party support, is a sign of maturity and a sign the product is taken seriously.

3.2.6 Documentation

There are two main types of documentation (Erenkratz and Taylor, 2003):

- User documentation
- Developer documentation

User documentation contains all documents that describe how to use the system. For certain applications there can be different levels in the user documentation, corresponding with different user levels and rights. For example, many applications that have an administrator role, have a separate piece of documentation for administrators. Additionly, there can be various user-contributed tutorials and How-Tos, be it on the project's website or elsewhere.

The other main type of documentation, which plays a much larger role in Open Source software than in proprietary applications, is developer documentation. A voluntary decentralised distribution of labour could not work without it (Weber, 2004, p.79). The developer documentation concerns separate documents on how to add or change the code, as well as documentation within the source code, by way of comments. The comments usually explain what a section of code does, how to use and change it and why it works like it does. Though this type of documentation may exist for proprietary software, it is usually not public. have access to it.

A third type of documentation that is often available for larger server-based applications is maintainers documentation, which includes the install and upgrade instructions. These need to be clear, with the required infrastructure and the steps for installing the software properly explained.

Documentation is often lagging behind the status of the application, since especially user documentation is often written only after functionality is created (Scacchi, 2002).

[5] A bug tracker is an application, often web based, in which the users can report problems with the software, the developers can assign the bug to someone who will handle it, and the status of the bug can be maintained. Bugzilla is one such package that is often used for this purpose.

3.2.7 Security

Security in software, especially when discussing Open Source software, has two sides to it. There are people who believe 'security by obscurity' is better, meaning that the inner workings of the software are hidden by keeping it 'closed source'. Something which Open Source obviously does not do. The advocates of 'Security by Obscurity' see the openness of Open Source software as a security hazard. Others argue that the openness of Open Source actually makes it safer because vulnerabilities in the code are found sooner. Open Source software gives both attackers and defenders great power over system security (Cowan, 2003; Hoepman and Jacobs, 2005).

Security depends strongly on how much attention the developers give to it. The quality of the code has much to do with it, and that goes for both proprietary and Open Source software. If the code of proprietary software is not secure, the vulnerabilities may still be found. There are plenty of examples where this occurs such as the Microsoft Windows operating system. The vulnerabilities are often found by 'hackers' who try to break the software, sometimes by blunt force or simple trial and error. In this case a vulnerability might get exploited before the vendor knows about it. The attack is the first clue in that case. The Open Source software's vulnerabilities, however, could be found by one of the developers or users, just by reviewing the code, and report the problem, so it can be fixed (Payne, 2002).

It is important that the developers take the security of their software seriously and respond swiftly to any reported vulnerabilities.

3.2.8 Functionality

Though functionality comparison is not specific to Open Source software evaluation and is properly covered in most traditional software evaluation models, there are some points to take into consideration. Open Source software often uses the method 'Release Early and Often' (Raymond, 1998b). This method enables faster error correction (Weber, 2004, p.80), by keeping the software up to date as much as possible. It also encourages people to contribute because they see the result of their work in the next release much sooner (Raymond, 1998b). However, this often means that the software is incomplete during the first releases, at least more so than is customary with proprietary software.

Where vendors of proprietary software will offer full functionality descriptions for their software, Open Source projects might not have the complete information on the website (Golden, 2005, p100-101). Just like with documentation, the information on the website might be lagging behind the actual functionality. Other means of checking the current functionality set might be needed. Fortunately, Open Source software that is freely available gives the added option of installing the software which enables the full testing of the functionality, an option that is mostly not available with proprietary software, where at most only limited versions, in terms of functionality or time, are given freely for trying out the software.

3.2.9 Integration

Duijnhouwer and Widdows (2003) mention three integration criteria. These are most important for software that is being used in collaboration with other software, and

for those who are planning on adapting the software to their use, such as adding functionality or customising certain aspects so that it fits better in the organisation's environment.

Modularity

Modularity of software means that the software or part of the software is broken into separate pieces, each with their own function. This type of structure has the following advantages:

- Modular software is easier to manage (Mockus et al., 2002; Garzarelli, 2002), and with a base structure that handles the modules well, people can easily add customised functionality without touching the core software. This way the software can still be upgraded without losing any custom code, because the module works alongside the core software which need not be changed. It also encourages contribution to the software. When someone writes a new module for his own use he may contribute this module back to the community later so others can use it too.
- Modular software enables the selection of the needed functionality, leaving out those that are not necessary for the intended use. This way the software can be customised without the need for a programmer.
- Use in commercial software – by making software modular, not everything needs to be given away as Open Source. It is can be used to give away only parts of software as Open Source while the add-on modules are sold as proprietary software (Duijnhouwer and Widdows, 2003), also called the 'razor' model – giving away the razor for free and charging for the blade (Golden, 2005, p.34).

Standards

In the software market more and more open standards emerge to make cooperation between software easier (Golden, 2005, p.190). If the software vendors use these standards in their software, it makes it easier to communicate between different software packages, and to switch between software packages. In some industries standards are far more important than in others. For some software there may not even be an applicable standard.

Well known examples of standards are the HTML and XML standards as defined by the World Wide Web consortium[6]. The XML standard in particular has grown to a widespread solution for integration.

The use of current and open standards in Open Source software is a sign of the software's maturity (Duijnhouwer and Widdows, 2003).

Collaboration with other products

Closely connected to standards is the collaboration with other products. As mentioned before, not every software type has applicable standards, and sometimes the

[6] W3c, `http://www.w3c.org`

formal standards are not used as much as other formats. For example, the Microsoft Word document format, the .doc, is probably the most used for exchanging editable documents. Software such as OpenOffice[7] has the ability to import and export to the .doc format, enabling its users to exchange documents with Microsoft Word users (Varian and Varian, 2003; OpenOffice.org, 2005).

Another example is the document preparation system LaTeX that has add-ons that allow the generation of PDF[8] versions of scientific documents. This thesis was also constructed using that system.

Software Requirements Most software is written for a specific Operating System (OS), for example Microsoft Windows or Linux (Wheeler, 2005). Certain types of software also rely on other software, such as a web server or a database. The requirements of the software will state which software and which versions of that software are compatible. If these requirements are very specific it could lead to problems if they are incompatible with the organisation's current environment.

3.2.10 Goal and origin

'Every good work of software starts by scratching a developer's personal itch' Raymond (1998b). This quote is about the motivation of Open Source software developers. Open Source projects get started for a number of reasons but the initial motivation has a clear goal of use for those who started the project.

How a software project started provides information on it's general usability. If the project started as a very simple small solution to the developer's problem, and has quickly grown into something much larger, the software might not be the most stable because it was not designed from the start the way it has become. If, however, the project was planned from the started to be something more substantial, the whole process is probably more thought through. This can be seen among other things in its use of modularity and standards, as mentioned above. A project's history or other documentation, such as the first release notes or community posts, can give information on how the project got started and the goal of the developer(s).

3.3 Selection

When evaluating software, a small list of candidates is needed which is evaluated fully. In order to get this list, a selection is performed on the list of all possible candidates to get the 'short list' (Golden, 2005, p.96-87).

There are five sources that can be used to populate a candidate list (Golden, 2005, p.94-96):

- Search Open Source Project Portals – for example SourceForge[9] and FreshMeat[10]

[7] `http://www.openoffice.org`
[8] A widely used cross-platform format for non-editable documents, created by Adobe
[9] `http://www.sourceforge.net`
[10] `http://www.freshmeat.net`

- Search the Web – this can also give pages about projects and user opinions
- Ask Open Source Developers
- Post to Mailing Lists
- Ask Vendors

3.3.1 The Selection Method

Anderson (1990) suggests five models for software selection. These are divided by compensatory and non-compensatory models. The simplest compensatory model is the Linear Weighted Attribute Model. In this model a number of attributes are used and each package gets a performance rating for each attribute. Weights are assigned to the attributes, which defines the compensatory nature of this model. The final score of each package is defined by the equation:

$$Q_i = \sum_{j=1}^{m} W_j A_{ij}$$

Where Q_i is the score of package i, W_j is the weight assigned to criterion j and $A_i j$ is the score of package i for criterion j. Thus the final score for a package is the sum of the weighted performance scores.

This method is suitable both for a quick selection and the complete evaluation process.

One of the other models is Elimination By Aspects (EBA). This model ranks the attributes by importance in descending order, and sets a minimum value for each attribute. Packages that do not conform to the minimum of the first attribute are eliminated, the remainder is tested against the second attribute's minimum, and so on.

Using the EBA model can save some time in the selection process. For example, a project that is clearly incomplete, or has not released an update in a long time, can easily be eliminated using EBA. However, in this step of the model the goal is to establish a short list for the in depth evaluation. A ranked list would be useful for this. In order to save time using elimination and still have a ranked list of suitable candidates as a result, the two methods are combined such that elimination is performed first, using one or more criteria, and the remainder of the candidates is ranked using the Linear Weighted Attribute Model.

3.3.2 The Criteria

In order to use these models a number of criteria are needed that give a good impression of the overall project and are reasonably easy to measure. The criteria suitable for this process are:

- Functionality
- Community
- Release activity
- Longevity

The first criterion is functionality. Though this criterion was not mentioned as the first criterion for Open Source evaluation, this is a criterion needed in the selection process in order to screen out any software that is clearly incomplete or does not fulfil the functional requirements. The other criteria are the first three of the full evaluation model and give a good impression of the level of activity of a project.

Each of these criteria is checked by visiting the project's website. The information needed may have to be searched for, in other cases it is in an obvious place. The more visible the information is, the better. If a website is hard to navigate it will keep others away as well, which can cause lower activity.

When combining EBA with the Linear Weighted Attribute Model, a minimum requirement is needed for the criteria used in the EBA step. Two of the criteria are well applicable for this process: Functionality and Release Activity. For functionality a set of minimum functional requirements can be used to eliminate the incomplete products. The release activity can have a minimum period since the last release, so that outdated projects or projects that have ceased development can be eliminated.

Functionality

The software's functionality can be found:

- In a feature list on the website (Golden, 2005, p.100)
- In an online demo of the software for web applications

If these two are not available or sufficient to determine the available functionality, the software can be installed to determine the functionality. This takes more time and is therefore better left to the evaluation step though.

Because of the Open Source 'Release Early and Often'(Raymond, 1998b) method there may be projects where the feature set is incomplete. Compare the functionality with the basic feature set of the software type or the functional requirements. If there are too many missing features, the software should not be taken into consideration, even if the features are said to be implemented in the near future. Base the selection on what is available instead of what is promised (Duijnhouwer and Widdows, 2003).

Community

As mentioned, community is a very important part of an Open Source project. The community's activity is expressed mostly through discussions in forums or mailing lists.

First thing to observe is the communities visibility. Is it clearly present on the projects website, is the visitor enticed to get into the community discussions? The better able one is to find the community and the more the visitor is invited to get involved, the more potential the community has for growth.

The activity level is observable by number of posts, the range of dates of the latest posts, etc. (Bergquist and Ljungberg, 2001). Most forums show the number of posts and the date of the latest post in each sub forum. In Appendix B two screenshots are given, one is an example of an active community, the other of a much less active one. Find the sub forum that deals with questions and/or problems (or the most general

one if there are more in that category), and check the dates of the top posts. The example where the activity is rather low shows that within the first 15 topics the date goes back six months, while the first 15 topics in the active community's sub forum go back only about 15 hours.

Keep in mind when evaluating the activity level, that a good community provides answers to questions, and feedback to suggestions, etcetera. This means that the higher number of posts in a topic, the better. If a question forum is filled with unanswered posts, so that the number of posts is close to the number of topics, it is not a very valuable community (Golden, 2005, p.140) & (Wheeler, 2005).

Release activity

Though community activity is very important, on the surface the developers' activity in the community is not always clearly visible. To quickly asses whether the developers are active, the releases of the software are investigated.

Release activity is measured by checking the releases made per period of time. This can usually be found in the download area or in the change log or release notes (Chavan, 2005), a page or text file that explains what has changed in a release since the last version. Sometimes change logs only list the last release, sometimes they list all the releases in descending order.

The change log also gives information that can help determine the significance of a release: How much has changed? Has functionality been added? The Open Source philosophy of 'Release Early & Often' (Raymond, 1998b) sometimes leads to very small changes between releases, where the frequency may be high, but development is not really that fast.

Longevity

Longevity is checked using the age (Golden, 2005, p.103), established using the date of the first release of the software. This can usually be found in the change log or the history of the project. When a software project stabilises over time the activity in terms of releases and community may eventually get lower. This happens in projects that are at least five to ten years old and have completed most functionality. The activity then is limited to bug fixing and perhaps a few ideas here and there about new or changed functionality (Wheeler, 2005). The age of a project can compensate for this decrease of activity. Giving age a significantly lower weight than the other criteria is recommended, because it is merely compensating for a small decrease in activity and is not a guarantee of stability.

3.4 Evaluation

The selection step should provide a small number of candidates to evaluate further. The evaluation step is done by evaluating each criterion for the shortlisted systems. Each candidate gets a relative score for each criterion. The way this score can be established is described per criterion in this section.

The final result is calculated as using the Linear Weighted Attribute Model[11] (Anderson, 1990). An example of the weights that can be used can be found in Chapter 4.

3.4.1 Community

The community is the driving force behind an Open Source project. The community defines much of the activity and reflects in other areas such as support and documentation.

The community is mostly visible in terms of (Duijnhouwer and Widdows, 2003; Crowston et al., 2004; Nijdam, 2003; Golden, 2005):

- Posts – number of posts per period, number of topics.
- Users – number of users, and the user/developer ratio in terms of the number of people and number of posts. If only users post, the developers are not as involved as they should be.
- Response time – if and how soon user questions are answered.
- Quality – the quality of posts and replies – are questions answered to the point, are the answers very short or more elaborate? Is there much discussion about changes and feature additions?
- Friendliness – how friendly the community is towards each other, especially to newcomers, also known as 'newbies'. The community should have an open feel to it, encouraging people to participate.

The depth of conversations as mentioned in the fourth item gives a good impression of how involved the community is with the ongoing development of the project. Much discussion about the software, in a friendly and constructive manner, encourages the developers to enhance the software further.

Another thing to check is whether or not all posts are kept or archived. When old posts in a community are being 'cleaned up', this can give considerable data loss that has important community value. Try to see if the community posts seem complete compared to the age of the project.

The community activity is also reflected in other areas such as support and documentation, but these are measured in the other criteria.

3.4.2 Release Activity

Release activity reflects the development progress. This is measured using the release frequency and significance per release.

Check the project's change logs to check (Chavan, 2005):

- The number of releases made per period of time – most projects will make several releases in a year, sometimes once or twice a month. A year is usually a good period to count the releases.

[11] See section 3.3.1 for the formula

- The significance of each release – the change log or release notes explain what has changed in the release. These descriptions are sometimes very elaborate, where every little detail is described, and sometimes very short, where just large changes are listed. A good distinction to make is whether the release only contains bug fixes or also contains enhancements to features or completely new features.

The project might also have a public roadmap. Check if any deadlines have expired and keep an eye on the roadmap as the project progresses to see if they keep to it.

If the project is listed on SourceForge and/or FreshMeat, some of the release activity information is available there.

3.4.3 Longevity

Longevity is a measurement of a project's stability.

Longevity is checked using (Golden, 2005, p.105) & (Nijdam, 2003):

- age of the product – the date of the first release
- version number – a 0.x number usually means the developers do not think the software complete or ready for production use at this time.
- if the project is very old it is worthwhile to check if it has gone through a cycle of redesign, or it is currently having problems with new technology.

Keep in mind that the version number doesn't always tell the whole story. Some projects might go from 1.0 to 2.0 with the same amount of change that another project has to go from 1.0 to 1.1. The fast progression of the version number might be used to create a false sense of progress. Other software products are still in a 0.x version, even after a long time, and after they are proved suitable for production use (Nijdam, 2003).

3.4.4 License

Check whether the license is an OSI approved license or, if they use a different license, read it carefully. If it uses one of the public licenses, the better known the license, the more can be found on its use and potential issues (Wheeler, 2005). Check if the license fits with the intended use.

3.4.5 Support

Support for Open Source software is in most cases handled by the community. The community's support areas are invaluable resources for solving problems (Golden, 2005, p.130). Mature products often have paid support options as well if more help or the security of a support contract is required.

Community support

The usage support is usually found in the community. Things to look for (Golden, 2005, p.137-141):

- Do they have a separate forum or group for asking usage related questions,
- How active is this forum,
- Are developers participating
- Are questions answered adequately

Check if responses to questions are to the point and if the responders are trying to be friendly and helpful. In the process of evaluating software, the evaluator will probably be able to post a question. Try to keep to the etiquette, where the most important rule is to search for a possible answer posting a question and to given enough relevant information for others to reproduce the problem (Golden, 2005, p.104) & (Wheeler, 2005).

The way the community is organised influences the community support's effectivity. A large project should have multiple areas for each part of the project, but the areas should not be spread to thin. That way the developers that are responsible for a certain part of the project are able to focus on the relevant area without getting overwhelmed with a large amount of other questions. If the areas are too specialised and little activity takes place in each, not enough people will show interest and questions are more likely to remain unanswered.

Failure support within the project is often handled by a bug tracker where problems are reported and tracked. Check how many of the developers are involved in this. Statistical studies have shown that in successful projects the number of developers that fix bugs in Open Source software is usually much higher than the number of developers creating new code (Mockus et al., 2000).

Paid support

Paid support might be available from the project team itself (Golden, 2005, p.135). Check the details and see if there are any people who have given their opinion about the quality of this support.

One of the strong signs of maturity of Open Source software is the availability of third party support: companies that offer commercial support services for Open Source products (Duijnhouwer and Widdows, 2003). Check if any third party support is available and if the form of support they offer is useful. Some companies offer service contracts, others offer only phone support on a per-incident basis.

Check for paid support options whether they will be used or not (Duijnhouwer and Widdows, 2003). How the situation may be during actual use of the software is not always clear and it can give a better impression of the maturity of the software.

3.4.6 Documentation

Documentation involves any text made available to help with the installation, use and development the software. User documentation and developer documentation

are the two main areas. A third area is maintainer's documentation – the install and upgrade instructions. See if these are available either on the website and/or packaged with the software and if these documents are clear.

User Documentation

Check for the existence of user documentation on the project's site. The site will in most cases have a separate section for documentation. The minimum that is usually available are instructions on the installation of the software. Additional documentation may include descriptions of the main features, 'How-Tos' and/or tutorials with instructions (Duijnhouwer and Widdows, 2003).

If the software has different access levels, such as administrator and normal user, see if this distinction is made in the documentation as well.

The documentation for larger projects is often handled by a documentation team. See if there is a discussion area about the documentation and how active they are.

Developer Documentation

The developer documentation consists of

- documents that describe the development process and how to participate
- comments in the source code that explain what the file or portion of code does and why

Check the available developer documents if they are clear on how to develop the software (Wheeler, 2005) and how to join the developer community.

Check the code whether a file starts with a short description of the file's use, and if there are any comments made throughout the file and how clear they explain the how and why of the file's operations, and whether the comments match the workings of the code.

A programmer or at least someone with some experience in programming will be better able to evaluate whether this documentation is set up well, especially the comments in the source code. It is a good idea to let someone with experience take a look at this documentation (Wheeler, 2005).

3.4.7 Security

There are various security advisories to check for bugs in all types of software that make it vulnerable to attacks. A couple of well known advisories are `http://www.securityfocus.com` and `http://www.secunia.com`. Check these sites for vulnerabilities in the software and see how soon they were fixed in a new version. Keep in mind that more popular software will have a higher chance of having vulnerability reports, so the mere lack of reports is no proof of its security.

On the project's website it can be seen, for instance in the community or in release notes, how serious the project is about security.

3.4.8 Functionality

One problem with Open Source projects is that the documentation is not always up to date with the latest software. Look beyond the feature list on the website to find out what features the software has. Resources for this are (Golden, 2005, p.99-102): querying the developers and asking the user community

Eventually the software itself should be investigated. If it is a web-based application, an online demo might be available, though installing it on a test environment could be useful because it also gives insight on how well the software installs.

A list of functional requirements for the goal of use of the software can be used to check if the needed functionality is available. If such a list is not given, there may be one available from technology analyst organisations (Golden, 2005, p.93). It is wise to make a distinction in the list between features that are absolutely necessary, where the absence would lead to elimination, and those that would be a plus, which results in a higher score. If there is something missing there is always the option to built it or have it built.

When comparing functionality, those features that are part of the functional requirements should take priority, but additional features may prove useful later. The features used or requested by the users in the future is not really predictable. While evaluating the software, features may be found in some of the candidates that are very useful for the goal. These can be added to the functional requirements.

Part of the functionality is localisation. The languages in which the interface and documentation are translated are a sign of the global interest taken in the software.

3.4.9 Integration

Modularity

Check the documentation and code base for evidence of a modular structure. The development documentation describes how to add modules if the software is modular.

Standards

Software projects that use standards usually include information about this in their feature list or documentation. Check whether the standards implementation is complete or still underway.

Collaboration with other products

If the software can work with relevant other product this can usually be found in the feature list or documentation.

Software Requirements Check the software requirements in the documentation of the software. There is usually a section on this in the installation document. Check whether these requirements can be met in the current environment (Wheeler, 2005), and how flexible they are in terms of the versions of the required software.

Compatibility with new versions of the required software might be an issue. Check how fast the software catches up with changes in the required software.

3.4.10 Goal and Origin

Something about the history of the project can often be found in the documentation or on a special page on the website. See if the goal of the software and the reason it got started is compatible with the intended use.

3.5 Model overview

The criteria and the evaluation process were described in the previous sections. To conclude this chapter, an overview of the model is given in two tables. Table 3.2 gives the background information on the criteria and Table 3.3 gives the selection and evaluation process for each criterion.

Table 3.2: Criteria Overview

Criterion	Background
Community	The driving force and main resource of an Open Source project.
Release activity	Shows development activity and progress.
Longevity	Indication of stability and chance of survival.
License	Public or specialised? GNU GPL is a well known public copyleft license. Copyleft ensures code and derivatives stay under that license
Support	Community and paid support, answering questions and failure support.
Documentation	User manuals and tutorials – developer documentation about code structure and coding guidelines.
Security	Openness vs. obscurity. Security needs to be taken seriously.
Functionality	Release early and often can lead to incomplete products. Feature information not always clear.
Integration	Modularity, standards and collaboration with other products.
Goal and origin	Does the project team's goal fit with the intended use? Gives an indication of how serious the developers are about the project.

Table 3.3: Evaluation Model Overview

Criterion	Selection	Evaluation
Community	Number of posts and users.	Number of posts, total and per period, number of users and developers, speed and quality of replies, community contributions (code, documentation, etc).
Release activity	Release count and significance.	Releases per period and significance, roadmap.
Longevity	Date since first release.	Age, version, gone through redesign cycles? Version number is no guarantee.
License		Public license i.e. GNU GPL? Does the license fit with goal of use?
Support		Community activity and quality, paid options from vendor and/or third party?
Documentation		User and developer documentation. Quality? Is it up to date?
Security		Security advisories and bugtracker. Response time to security holes?
Functionality	Check with minimal requirements using feature list, querying developers and community, demo's for web based apps.	Check feature list, ask developers, online demo, test by installing. Compare with functional requirements. Check richness of features, additional features that may be useful, these give an indication of target audience and enthusiasm of developers.
Integration		Check the code structure and documentation for modularity. Are relevant standards implemented? Does it work with relevant products, how flexible is it with required software versions?
Goal and origin		Check documentation for history and roadmap.

Chapter 4

Case Study: Course Management Systems

Now that the model for Open Source software evaluation has been defined, it needs to be tested on real software systems. Course Management Systems were chosen as the target software category, because this relatively new area of software where Open Source alternatives are now starting to break through, is not explored fully at this time, and it concerns a reasonably large sized piece of software. The target users for these systems are educational institutions, including universities. These institutions make long term investments in this type of software which deserves a full evaluation process.

First, an introduction will be given on these systems. Then the model is applied to a candidate list to find the highest scoring systems, and these results are compared to their real life performance.

4.1 Introduction

Many universities, including Tilburg University, have been using a Course Management System (CMS) for a few years now. A Course Management System is a web based application through which students and teachers can interact. Course documents such as presentation sheets and assignments are made available to the students by the teacher and students can work on assignments in groups and take online tests.

There is no agreed upon definition or even one single term for CMSs. In the United Kingdom they are often called Virtual Learning Environments (VLEs). Other terms used are Managed Learning Environment (MLE), Learning Management System (LMS) and Learning Support System (LSS) (Wikipedia, 2005b).

In this case study, Open Source Course Management Systems are investigated. In software evaluation there usually is a list of functional requirements. However, because there is no clear definition of a CMS and its most common properties, and there is no outside functional requirements list available in the context of this thesis, some assumptions are needed. A list of functional requirements was created using

the main features from an online CMS overview: Edutools (2005a) and a report comparing a large number of Open Source CMSs: COL (2003). This list can be found in Appendix C.

In the Netherlands the universities mainly use the proprietary CMS Blackboard[1] (Surf, 2005a). This system is the current market leader along with another proprietary system called WebCT [2]. In Appendix D the use of CMSs in the Netherlands is given in detail.

The CMS Open Source market is relatively young. There are a few promising candidates and these are getting more attention.

A search is performed to make a list of candidates to perform the selection process on. After the selection process the top systems are evaluated. This was restricted to two systems because of time constraints.

The full candidate list was formed by investigating other comparison articles and doing web searches. Two resources that gave extensive lists of candidates are the Commonwealth of Learning's LMS Open Source Report of 2003 (COL, 2003) and WCET's Edutools.info website, which provides a CMS comparison tool, using detailed descriptions of functionality and background information of proprietary as well as Open Source CMSs (Edutools, 2005a). The candidate list that was constructed using these two sources has 36 candidates. The complete list with overview of the sources and links to the project's website is included in Appendix E.

4.2 Selection

Because there are some candidates that are very limited in terms of completeness and activity, Elimination By Aspects (EBA) is performed to eliminate that part of the candidates before ranking the remaining candidates by score using the Linear Weighted Attribute Model (Anderson, 1990).

4.2.1 Elimination By Aspects

The first step uses Elimination By Aspects (EBA) to eliminate those candidates that do not conform to a minimum standard. Two criteria are applied in the EBA step, using the following minimum values:

- **Functionality:** The software needs to have the basic functionality to conform to the functional requirements that were defined.
- **Release Activity:** The last stable release needs to be no older than two years.

Furthermore, any projects that have an announcement of discontinuation on their website are eliminated. Some of the software turned out not to be Open Source. These candidates were also eliminated.

The website is checked, if it is not available, a web search for a possible new address could lead to the current site. Information on the last release and the description of

[1] `http://www.blackboard.com/`
[2] `http://www.webct.com`

the features are checked. If the available information does not convince that it does not meet the requirements the project will not be eliminated, so that only candidates that do not meet the requirements for certain are excluded.

The elimination step reduced the list from 36 to 19 candidates that will be evaluated in the scoring step of the selection process. The details can be found in Appendix F.

4.2.2 Ranking

Using the result from the elimination, the remaining 19 candidates are scored using the Linear Weighted Attribute Model. Relative scores from 1 to 10 are assigned for each criterion. Information such as dates and numbers, as well as remarks relevant to the determined score are summarised in tables in Appendix F.

Weights

As discussed before, age is a compensation for relatively small differences in activity levels. Therefore, the longevity weight should be significantly lower than the weight of the other criteria.

The weights can be set according to the situation and preference of the evaluating party. The optimal distribution of the weights was not investigated in the context of this thesis. The choice was made to distribute the weights in such a manner that the total maximum score, given that scores could range from 1 to 10, would be 1000, so the sum of the weights should be 100. Age was given a weight of 10 and the remaining 90 points were distributed evenly by level of importance, resulting in the following weight distribution:

- Functionality – 35
- Community – 30
- Release Activity – 25
- Longevity – 10

Because of the manner in which this weight distribution was chosen, the results will be tested by checking the final results with different weight distributions to see if the results remain consistent.

There is certainly room for improvement in this process. Using statistical analysis could lead to more precise conclusions. However, the results established here gives a reasonably trustworthy indication.

The results for each criterion are presented in a table in Appendix F. Below is a short description of the process.

Functionality

In order to check the functionality quickly for each candidate, the functionality requirements are compared to the feature lists and if available the online demo. This criterion is hard to measure, so the score is harder to determine. In this case a coarser

scale, using the values 2, 4, 6, 8 or 10 for the score, is used. Determining these scores is a rather intuitive process, left to the user's situation.

The demos were used where possible, but unfortunately, not all projects had an online demo available. The feature lists are not always accurate, so care was taken to try and determine how well they represented the actual state of the software. The functionality will of course be checked further for the shortlisted candidates in the evaluation.

In this case, some help came from the edutools website, which gives detailed comparisons of features by giving a description of the feature possibilities for each product. Twelve of the nineteen CMSs can be found on this site.[3] Unfortunately, some of the information is rather dated – for example, the ILIAS version presented here is almost two years old while the last release is just a month old – so for some products, the recent changes have to be taken into account compared to this information. For the remainder of the CMSs the resources of their own websites need to be used.

Community

For community scoring, the accessibility is the first point to evaluate. Then the number of topics and/or posts in the community message boards, forums or mailing lists are counted. This can be challenging at times because not all message services provide total counts, and some may only provide the total number of messages, but not the number of topics, and vice versa.

The activity between the projects was compared and scored from 1 to 10, assigning identical scores for activity that was very close together. Because there were not always totals available, it could not be used as a direct measure.

Release activity

The release activity was scored by using three values

- latest release date
- total number of releases in 2004 and 2005
- weight of the release – a number from 1 to 10 indicating the average significance of the releases

The following equation was used to determine the score. The relative score of the age of the last release, 10 being the product with the most recent last release and 1 being the product with the oldest:

$$S_{1i} = 10 \times (1 - (D_i / \max D))$$

Where D_i is the number of days since the last release of candidate i and $\max D$ is the maximum value of number of days since the last release over all candidates.

The relative score of the number of releases multiplied by the release weight:

$$S_{2i} = 10 \times ((R_i \times W_i) / \max (R \times W))$$

[3]The side-by-side comparison of these twelve can be found here `http://www.edutools.info/course/compare/compare.jsp?product=215,255,239,218,152,217,234,74,165,203,183,162`

Where R_i is the number of releases for package i, W_i is the average weight per release for package i and $\max(R \times W)$ is the maximum of all values $R_i \times W_i$ over all candidates.

$$\text{Total Score} \quad T_i = (S_{1i} + (3 \times S_{2i}))/4$$

The S_{2i}, the number of releases and their weight is given a higher importance than S_{1i}, age since the last release. A project could release a new version tomorrow that could lead to large shifts in those numbers, whereas the number of releases in a recent period combined with their weight, gives a better impression of the overall progress. Therefore the latter makes up $3/4$ of the score.

This method is not validated statistically and could be improved upon. One problem with this equation is that there is no compensation for the correlation between number of releases and last release date that inevitably exists when the last release was for example in early 2004, and thus the number of releases will be low as well.

Longevity

As compensation for decreasing activity over time both in releases and the community, the age of a project is measured. For this the inverse of the last release formula was used, so instead of the smaller number of days scoring higher, the larger number of days scores higher:

$$S_i = 10 * (D_i / \max D)$$

Where D_i is the number of days since the first release of candidate i and $\max D$ is the maximum value of number of days since the first release over all candidates.

4.2.3 Scoring Result

The final result for each candidate is calculated using the Linear Weighted Attribute Model equation:

$$Q_i = \sum_{j=1}^{m} W_i A_{ij} \qquad \text{(Anderson, 1990)}$$

The result of the scoring step leads to a list of total scores ranging from 960 to 135 out of a possible 1000-100. The top five candidates are:

- Moodle – 960
- ATutor – 800
- Claroline – 775
- ILIAS – 770
- Dokeos – 675

The full results are given in Appendix F.

When looking at a chart of the scores per criterion in Appendix F.7, it can be seen that the criteria tend to show the same decrease down the list as the total scores, only age is more varying across the ranked list. As explained before age is only a small compensation for a decrease in activity in the other criteria. Because the three most important criteria show mostly the same line of decrease as the total score,

changing the weights of these criteria did not cause any significant differences in the end result. The top five in almost all cases stayed the way it is.

One note on Dokeos is in order. Dokeos is a project that has split off from Claroline in 2004, a development team is still working on each project. The software is still very close together. Dokeos scored lower than Claroline understandably on release activity and age.

The sixth candidate on the list scored 525 points. A significant difference with the number five, if the remarks concerning Claroline and Dokeos are taken into account. Therefore this list of five candidates would be a good list for the evaluation step. However, due to the limited time available for this research project, the evaluation step is performed on the top two candidates: Moodle and ATutor.

4.3 Evaluation

The selection step resulted in a list of candidates ranked by the selection score. The top two will now be evaluated: Moodle and ATutor.

Scores ranging from 1 to 10 are given on all criteria for each system. One is given when it does not fulfil any of the wanted characteristics of the criterion, and ten when that the software ideally complies with the criterion.

Before performing the evaluation some general information on both Moodle and ATutor will be given.

4.3.1 Introduction

Moodle

Moodle is the creation of Martin Dougiamas, PhD student at Curtin University, Perth, Australia. He was a WebCT administrator for Curtin University. Using this hands-on experience, he set up his PhD project, entitled:

'The use of Open Source software to support a social constructionist epistemology of teaching and learning within Internet-based communities of reflective inquiry'

Martin has co-authored several papers on the subject, together with Peter C. Taylor, Associate Professor at Curtin University (Dougiamas and Taylor, 2003).

This research project started in 2000. The first public release of Moodle, version 1.0, was released on August 20, 2002. In 2003 the company moodle.com was launched, offering professional support and management of Moodle installations. Currently, the project is at version 1.5, with a large number of features added since the first release. The project has grown substantially, with many users and developers participating in the community at Moodle.org (Moodle, 2005a).

Philosophy Moodle is based on the 'social constructionist pedagogy'. The page explaining the philosophy in the Moodle Documentation can be found in Appendix G.

This philosophy is centred around 4 terms (Moodle, 2005h):

- Constructivism – People actively construct new knowledge as they interact with their environment.
- Constructionism – Learning is particularly effective when constructing something for others to experience.
- Social Constructivism – Extending constructivism into a social group constructing things for one another
- Connected and Separate – Separate behaviour occurs when trying to remain 'objective' and 'factual'. Connected behaviour occurs when accepting subjectivity, trying to listen and ask questions to try and understand the other point of view. Constructed behaviour occurs when a person is sensitive to both of these approaches.

Moodle's infrastructure is supportive of this philosophy in particular but it does not force it upon the user (Moodle, 2005h).

ATutor

ATutor is an Open Source Web-based Learning Content Management System (LCMS) designed with accessibility and adaptability in mind. The team consists of twelve people. The project originated in Canada and has a few Canadian sponsors. ATutor complies with the W3C WCAG 1.0 accessibility specifications at the AA+ level, allowing access to all potential learners, instructors, and administrators, including those with disabilities who may be accessing the system using assistive technologies (ATutor, 2005c) & (ATutor, 2005d).

A six point model that draws on popular understanding of learning and the structure of knowledge is their starting point for developing an intelligent learning environment that adapts to all who use it. The full philosophy is included in Appendix H. (ATutor, 2005d)

1. Visual learners like to see or imagine things
2. Verbal learners like to hear or verbalize things
3. Kinesthetic learners like to do or experience things
4. Global learners structure information in webs
5. Hierarchical learners structure information in trees
6. Sequential learners structure information in chains

4.3.2 Evaluation Result

The evaluation as described in the model in Chapter 3 was applied to evaluate these two candidates. For each criterion a detailed description of the steps followed and the observations made can be found in Appendix I.

The following is a summary of the evaluation results.

Community

Moodle shows very high activity levels. Over 14000 topics in the most active course, 26–28 replies a day in the most active forum of that course. ATutor is by far not that active with 1140 topics and 3–4 replies a day. Moodle's discussions are often about the functionality of the product, actively discussing possibilities for additions and changes. ATutor's post consist mainly of short support questions and answers.

Release Activity

Moodle has a higher release activity than ATutor: 26 vs 14 releases in total.

Longevity

ATutor and Moodle both started in 2002, with version 1.0 released in the second half of that year.

New Technology Both products are based on PHP and MySQL, each of which has released significantly changed versions last year. Moodle appears to have no problems with the new versions, ATutor is still working on some issues.

License

Both products are licensed under the GNU GPL

Support

Community support for Moodle is more active than for ATutor. Quality level of the help offered is also higher. Paid support options are available for both but Moodle's options seem more elaborate.

Documentation

Both products have a documentation section which is reasonably complete. Moodle, however, has a number of elaborate user-contributed manuals and tutorials. Developer documentation is also available for both.

Security

Moodle seems to be more involved in security with an open bug tracker and fast responses to vulnerabilities.

Functionality

Both products have the basic functionality that were named in the requirements. However, Moodle has more additional features and all features seem much richer than ATutor's, with more options for customisation. Moodle's modular structure allows for more user-contributed additions.

Integration

Standards SCORM is a package of standards used with CMSs (Edutools, 2005b). Both systems support SCORM.

Modularity Moodle has a modular structure for activities, languages and themes. ATutor does not show signs of a modular structure.

Compatibility with other applications Moodle has more options for working with other applications, such as authentication protocols, than ATutor.

In terms of **Software Requirements**, Moodle is more flexible with the versions of the required software it works with than ATutor. Moodle's developer documentation has flexibility listed as a requirement: *'Moodle should run on the widest variety of platforms'* (Moodle, 2005c).

Goal and Origin

Moodle's origin is the PhD project Martin Dougiamas started and has a clear goal, which fits with university institution-wide use of the CMS. ATutor's origin is unclear, the only goal that is clearly stated is the compliance to accessibility standards.

4.3.3 Results

The evaluation shows a very conclusive result. The scores are shown in Table 4.1. In every respect Moodle scores higher than ATutor, except in License where they achieve equal scores. The weights used are also included in the table.

Table 4.1: CMS Evaluation Results

Criterion	**Weight**	**Moodle**	**ATutor**
Community	10	9	5
Release Activity	8	10	6
Longevity	6	10	9
License	4	10	10
Support	6	10	5
Documentation	6	9	7
Security	4	9	4
Functionality	8	9	6
Integration	6	9	5
Goal and Origin	2	10	5
Weighted Total		**9.4**	**6.1**

The weights can be adjusted according to preference and the relevance for the type of software. For example, integration gets a reasonably high weight here because this type of software has integration possibilities in a number of areas, such as the standard package SCORM and authentication protocols. In more complex software, documentation might get a higher weight. Keep in mind the general importance of certain criteria, such as community and release activity, when adjusting these weights.

Because the sum of these weights does not lead to a nice round number, the score was calculated using the average weighted total score to give a number from 1 to 10. The weighted average is not significantly different from the unweighted average, because the results are reasonably even.

Overall it is clear that Moodle is complying much more to the requirements of a good Open Source project that are set in this model than ATutor does. Moodle certainly had a very open and collaborative feel to it from the moment the project was first encountered during this research project, which is what is needed to run a successful Open Source project.

4.4 Case Study: Conclusion

In the previous chapter, a model was defined to use for Open Source software evaluation. In this chapter, the case study that was performed using that model was described. The goal of the case study was to see if the model is useable for real software and if the results it returned are consistent with real performance of the software.

The model was used to evaluate Course Management Systems, starting with 36 candidates, and finally evaluating two systems. These two showed significant differences which illustrated the importance of the community which' activity defines the value of other criteria like documentation and support. The model was well applicable to these systems, the needed values could be found using the guidelines of the model.

4.4.1 Validity of Results

Now that the results from the evaluation of the CMSs are known, it is necessary to check whether these results are reflected in the performance of these systems in real life. There are a number of sources that can be used for this, including other comparison reports, evaluations and implementations done by educational institutions, and any publicity the systems have received.

The Top 5

The top 5 systems were investigated for real life performance. Dokeos and Claroline were investigated together because the systems currently do not differ enough to warrant separate results. Table 4.2 gives a summary of the findings. The full remarks can be found in Appendix J.

Table 4.2: Real Life Performance of Top 5 systems

Source	**Dokeos Claroline**	**ILIAS**	**ATutor**	**Moodle**
COL (2003)	–	Recommended second	Recommended First	Shortlisted
Clements (2003)	Shortlisted	–	Shortlisted	Recommended
VUB (2004)	Chosen and implemented	–	–	Shortlisted
Ose.Nz.Org (2004)	–	Shortlisted	Shortlisted	Chosen and implemented
UniKöln (2005)	–	Implemented	–	–

The investigation of the validity shows that the results of the selection and evaluation performed in the case study provides an accurate overview of the leading systems.

Other systems

The top five systems appear to get the most coverage in the resources that were found. A couple of other systems are also mentioned often:

- LON-CAPA
- Bodington Commons
- DotLRN (or .LRN)
- Sakai

These four systems were the ones that followed the top five mentioned above in the ranked list of the selection results, so it would appear that the model gives an accurate representation of the worthwhile systems.

Note on Sakai Sakai is a project that has just started. It is run by a joined effort between several universities, and is the continuation of three other projects (Sakai, 2005). It has the attention of many because it is believed to have much potential. Currently it is still very young.

Chapter 5

Conclusion & Further Research Recommendations

This thesis studied the Open Source market from a software evaluation point of view in order to create a model for Open Source software evaluation including a case study to test this model on real software.

In this final chapter the results of this research are discussed by answering the research question and subquestions posed in the first chapter.

5.1 Research Results

In this thesis the unique characteristics of Open Source software were investigated to construct a model for software evaluation of Open Source systems. In the first chapter a number of questions were formed that served as a guideline for the research done in this thesis. The answers to these questions will now be given by briefly recapitulating the main issues addressed in this thesis.

The first question for which an answer was needed is:

Are there characteristics that make Open Source software unique that are relevant to software evaluation, and if so, what are they?

A number of unique characteristics were found in relation to Open Source software, among which the way the projects are community driven and the openness of information that allows for a more comprehensive evaluation of this type of software. This answers the first part of this question. The characteristics were defined in the criteria of the model for Open Source software evaluation, combining this answer with the answer of the second question:

Which criteria can be defined for Open Source software selection and evaluation?

The following ten criteria were found using literature on Open Source software:

Community The community of an Open Source software project is the driving force behind the project.

Release Activity The releases of an Open Source software project are an indication of the activity and the progress of a project.

Longevity How long a project has existed combined with a healthy level of activity is a measure of the chance of survival of a project.

License A commonly used license, like the GNU GPL, is preferable, and the license needs to fit with the intended use.

Support Support for Open Source projects is available from three main sources – the community, paid support by the project team and paid support offered by third parties.

Documentation Development documentation plays an important part in the quality of the source code.

Security Security is the subject of heated debate, bugs can be found more easily in the source code by developers as well as attackers. Security needs to be taken seriously by the project team.

Functionality Different ways of evaluating the functionality are available, including of course installing the software without restrictions.

Integration Standards, modularity and collaboration with other products. Relevance depends greatly on the type of software, weight should be set according to the relevance.

Goal and Origin The goal and original reason for creating the software gives a good impression of whether it fits with the intended use and how serious the project is undertaken. Information may not be available, lower weight than average.

The third question that was defined is: **What information is needed to score these criteria?**

Using the practical part of the evaluation model, this question can be answered. For each criterion a description of the evaluation process and the information necessary to establish a score can be found in this thesis.

Four of the criteria are selected for the selection of a 'short list' of systems for on depth evaluation. They are chosen for their importance and the ability to evaluate these criteria in a short amount of time. These criteria are: Functionality, Community, Release Activity and Longevity.

Selection can be done using two of choice models described by Anderson: The Elimination by Aspects model to eliminate part of the candidate list that does not meet a minimum requirement on functionality and release activity, and the Linear Weighted Attribute Model, using the four criteria mentioned above, to establish a ranked list of the remaining candidates.

The result of the selection step should be a 'short list' of candidates that are at the top of the list. These systems can then be evaluated in depth using the defined criteria.

Finally, the model needs to be tested by applying it to real systems. The software category of Course Management Systems (CMSs) was used in the case study.

The question that was leading the case study is:

How well does this model apply to evaluation of Course Management Systems?

Using the two models for selection mentioned above, the top two systems were found that were evaluated in depth.

The evaluation process was followed successfully. The model was well applicable to this type of software. The results were verified using information on the real life performance of the top 5 systems. The findings were consistent with the result of the evaluation.

Now the final question can be answered:

'Is it possible to define a software evaluation model specifically aimed at Open Source software, and can this model be applied to select a Course Management System?'

The answer to this question: Yes. The model was defined using various literature and was well applicable, the model was followed to come to a satisfactory result in the case of CMSs, which was in agreement with the actual performance of the systems.

5.2 Contribution

The contribution of this thesis is twofold. A contribution is made to the target audience, the ones who want to evaluate Open Source software. The other is the contribution to scientific research.

5.2.1 Target audience

The goal of this research project was to produce a model that gives those that want to evaluate Open Source software, but are not familiar with this type of software, insights into the characteristics, the development method and the project that runs the development. These insights can be used to determine the best choice of software from a list of candidates.

5.2.2 Scientific research

The field of Open Source software in scientific research is still small. With the growing interest of the business world in Open Source software, the scientific research is growing as well, though it is still behind in many respects. This thesis can add to this research field. It gives insights into Open Source from a business use perspective. The model suggested here could be improved upon, so further research, building on the model given here, can be done.

5.3 Recommendations for Further Research

This research project has attempted to answer some questions regarding Open Source software. This relatively new area does still leave much to explore. There are a number of ideas to extend on this research project, as well as new questions that could be answered.

5.3.1 The Model

The model created in this thesis project could use more research and fine tuning. The model defines criteria and the points of attention for these criteria, but it could still use a more structural and better defined way of defining a score for a certain piece of software. For example, a more statistical approach could be taken to define community scores based on number of posts, users, and so on.

This model takes a general approach in terms of the stake holders that could be involved in the deployment and use of the software, such as system administrators, managers and users. One could investigate further how each criterion and its result affect the different stake holders.

One other perspective that could use different approaches using this model, is the situation of use, meaning whether the system is being deployed as-is or will be highly customised. High customisation means higher priorities for modularity, (developer) documentation, and so on. It also warrants further investigation of the source code.

In order to check whether the model works, a case study was conducted on Course Management Systems. Of course one case study does not lead to a definitive answer. The model proposed here could certainly be tested more thoroughly to see how it holds up, and improved using the results of these tests and the considerations made above.

While in the final stages of writing this thesis, my attention was brought to a paper that has a very similar goal, entitled 'Business Readiness Rating for Open Source; A Proposed Open Standard to Facilitate Assessment and Adoption of Open Source Software'. This model was developed by Spikesource, Carnegie Mellon West and Intel (OpenBRR, 2005). The model shows many similarities to the model that is proposed here. It would be interesting to investigate how these models relate, how they differ and in what sense they could compliment each other. More information on this model and the related developments can be found at `http://www.openbrr.org`.

5.3.2 Open Source software in education

Several articles, among which 'Open Source Opens E-learning' (Coppola and Neelley, 2004), argue that Open Source is very suitable for use in higher education, because of tight budgets and the fact that educational institutions often have some good software engineers on their staff, among other things. The use of Open Source applications in higher education, like the CMSs investigated in the case study of this thesis, can be further investigated. It seems that though Open Source seems a logical choice for universities, not many have taken an interest in this software so far.

Bibliography

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

ATutor. ATutor Requirements, 2005a. URL `http://www.atutor.ca/atutor/docs/requirements.php`. Retrieved on June 8, 2005.

ATutor. ATutor Website - Translation, 2005b. URL `http://www.atutor.ca/atutor/translate/index.php`. Retrieved on June 8, 2005.

ATutor. ATutor Website, 2005c. URL `http://www.atutor.ca/`. Retrieved on July 1, 2005.

ATutor. Atutor Website : About, Philosophy, 2005d. URL `http://atutor.ca/philosophy.php`. Retrieved on December 1, 2004.

ATutor. ATutor Roadmap, 2005e. URL `http://www.atutor.ca/atutor/roadmap.php`. Retrieved on June 8, 2005.

D. Becker. California considers open-source shift, 2004. URL `http://news.com.com/2100-7344_3-5327581.html`. Retrieved on August 8, 2005.

M. Bergquist and J. Ljungberg. The power of gifts: organizing social relationships in open source communities. *Information Systems Journal*, 11(4):305–320, 2001.

S. Carless. Nokia and Apple Collaborate On Open Source Browser, 2005. URL `http://apple.slashdot.org/article.pl?sid=05/06/21/1837219`. Retrieved on August 8, 2005.

A. Chavan. Seven Criteria for Evaluating Open-Source Content Management Systems. Linux Journal Website, 2005. URL `http://www.linuxjournal.com/node/8301/`. Retrieved on August 9, 2005.

I. Clements. Virtual Learning Environment Comparison Report. Technical report, Progress Through Training, 2003. URL `http://www.pttsolutions.com/modules.php?op=modload\&name=News\&file=article\&sid=74`. Retrieved on August 8, 2005.

COL. COL LMS Open Source. Technical report, Commonwealth of Learning, 2003. URL `http://www.col.org/Consultancies/03LMSOpenSource.pdf`. Retrieved on August 8, 2005.

C. Coppola and E. Neelley. Open source - opens learning, Why open source makes sense for education. Technical report, R-Smart Group, 2004. URL `http://www.rsmart.com/assets/OpenSourceOpensLearningJuly2004.pdf`. Retrieved on March 30, 2005.

C. Cowan. Software Security for Open-Source Systems. *Security & Privacy Magazine, IEEE*, 1(1):38–45, 2003.

K. Crowston, H. Annabi, J. Howison, and C. Masango. Towards A Portfolio of FLOSS Project Success Measures. In *Collaboration, Conflict and Control: The 4th Workshop on Open Source Software Engineering, International Conference on Software Enginnering (ICSE 2004)*, pages 29–33, 2004. URL `http://opensource.ucc.ie/icse2004/Workshop_on_OSS_Engineering_2004.pdf`. Retrieved on March 30, 2005.

P. Donham. Ten Rules for Evaluating Open Source Software. Point of view paper, Collaborative consulting, 2004. URL `http://www.collaborative.ws/leadership.php?subsection=27`. Retrieved on August 8, 2005.

M. Dougiamas and P.C. Taylor. Moodle: Using Learning Communities to Create an Open Source Course Management System. In *Proceedings of ED-MEDIA 2003*, 2003. URL `http://dougiamas.com/writing/edmedia2003/`. Retrieved on March 30, 2005.

F. Duijnhouwer and C. Widdows. Capgemini Open Source Maturity Model. Website: http://www.capgemini.com/technology/opensource/solutions.shtml, august 2003. URL `http://www.seriouslyopen.org/nuke/html/modules/Downloads/osmm/GB_Expert_Letter_Open_Source_Maturity_`. Retrieved on March 30, 2005.

Edutools. Edutools Course Management Systems, 2005a. URL `http://www.edutools.info/course/`. Retrieved on August 8, 2005.

Edutools. Edutools Glossary - Instructional Standards Compliance, 2005b. URL `http://www.edutools.info/course/help/glossary.jsp#20`. Retrieved on August 13, 2005.

J.R. Erenkratz and R.N. Taylor. Supporting Distributed and Decentralized Projects: Drawing Lessons from the Open Source Community. Technical report, Institute for Software Research, 2003. URL `http://www.erenkrantz.com/Geeks/Research/Publications/Open-Source-Process-OSIC.pdf`. Retrieved on August 9, 2005.

FSF. The Free Software Definition, 2005a. URL `http://www.gnu.org/philosophy/free-sw.html`. Retrieved on August 8, 2005.

FSF. Why “Free Software” is better than “Open Source”, 2005b. URL `http://www.fsf.org/licensing/essays/free-software-for-freedom.html`. Retrieved on August 8, 2005.

G. Garzarelli. The Pure Convergence of Knowledge and Rights in Economic Organization: The Case of Open Source Software Development. In *Industrial Dynamics of the New and Old Economy - Who is Embracing Whom?*, 2002.

R.L. Glass. A Sociopolitical Look at Open Source. *Communications of the ACM*, 46 (11):21–23, 2003.

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

A. Hars and S. Ou. Working for Free? – Motivations of Participating in Open Source Projects. *International Journal of Electronic Commerce*, 6:25–39, 2002.

G. Hertel. Motivation of software developers in Open source projects: an Internet-based survey of contributors to the Linux kernel. *Research Policy*, 32(7):1159–1177, 2003.

J. Hoepman and B. Jacobs. Software Security Through Open Source. Technical report, Institute for Computing and Information Sciences, Radboud University Nijmegen, 2005. URL `http://www.cs.ru.nl/~jhh/publications/oss-acm.pdf`. Retrieved on August 9, 2005.

ILIAS. ILIAS website, 2005. URL `http://www.ilias.de`. Retrieved on August 2, 2005.

B. McMullin and M. Munro. Moodle at DCU, 2004. URL `http://odtl.dcu.ie/wp/2004/odtl-2004-01.html`. Retrieved on August 8, 2005.

A. Mockus, R.T. Fielding, and J. Herbsleb. A Case Study of Open Source Software Development: The Apache Server. In *Proceedings of the 22nd International Conference on Software Engineering (ICSE 2000)*, 2000. URL `http://opensource.mit.edu/papers/mockusapache.pdf`. Retrieved on March 30, 2005.

A. Mockus, R.T. Fielding, and J. Herbsleb. Two Case Studies of Open Source Software Development: Apache and Mozilla. *ACM Transactions on Software Engineering and Methodology*, 11(3):309–346, 2002. URL `http://opensource.mit.edu/papers/mockusapache.pdf`. Retrieved on March 30, 2005.

Moodle. Moodle.org - Background, 2005a. URL `http://moodle.org/doc/?frame=background.html`. Retrieved on August 8, 2005.

Moodle. Moodle Buzz - reviews, papers and listings about Moodle, 2005b. URL `http://moodle.org/mod/resource/view.php?id=102`. Retrieved on August 8, 2005.

Moodle. Moodle Developer Manual, 2005c. URL `http://moodle.org/doc/?file=developer.html#architecture`. Retrieved on August 13, 2005.

Moodle. Course: Moodle Documentation, 2005d. URL `http://moodle.org/course/view.php?id=29`. Retrieved on August 8, 2005.

Moodle. Moodle Documentation - Installation Requirements, 2005e. URL `http://moodle.org/mod/resource/view.php?id=3856`. Retrieved on August 8, 2005.

Moodle. Moodle Downloads - Language Packs, 2005f. URL `http://download.moodle.org/lang`. Retrieved on August 8, 2005.

Moodle. Moodle Downloads - Activity Modules, 2005g. URL `http://download.moodle.org/modules`. Retrieved on August 8, 2005.

Moodle. Moodle Philosophy, 2005h. URL `http://moodle.org/doc/?frame=philosophy.html`. Retrieved on August 13, 2005.

Moodle. Moodle Roadmap, 2005i. URL `http://moodle.org/doc/future.html`. Retrieved on August 8, 2005.

moodle.com, 2005. URL http://www.moodle.com/. Retrieved on June 7, 2005.

Mozilla. Mozilla.org Support, 2005. URL http://www.mozilla.org/support/. Retrieved on February 16, 2005.

MySQL. MySQL Support Website, 2005. URL http://www.mysql.com/support/premier.html. Retrieved on February 16, 2005.

Netcraft. July 2005 Web Server Survey, 2005. URL http://news.netcraft.com/archives/web_server_survey.html. Retrieved on July 1, 2005.

M. Nijdam. Vijf adviezen voor selectie van oss-compontenten. *Informatie : maandblad voor informatieverwerking*, 45(7):28–30, 2003.

Novell. Novell Announces Agreement to Acquire Leading Enterprise Linux Technology Company SUSE LINUX, 11 2003. URL http://www.novell.com/news/press/archive/2003/11/pr03069.html. Retrieved on August 8, 2005.

OpenBRR. Business Readiness Rating for Open source; A Proposed Open Standard to Facilitate Assesment and Adoption of Open Source Software, RFC1, 2005. URL http://www.openbrr.org/docs/BRR_whitepaper_2005RFC1.pdf. Retrieved on August 10, 2005.

OpenOffice.org. OpenOffice.org Writer Product Information, 2005. URL http://www.openoffice.org/product/writer.html. Retrieved on August 10, 2005.

OReilly. Online Catalog - Using Moodle, 2005. URL http://www.oreilly.com/catalog/moodle/. Retrieved on August 8, 2005.

T. O'Reilly. Ten Myths about Open Source Software. URL http://opensource.oreilly.com/news/myths_1199.html. Retrieved on August 8, 2005, Published on O'Reilly (http://www.oreilly.com), 1999.

Ose.Nz.Org. LMS Technical Evaluation, 2004. URL http://eduforge.org/docman/view.php/7/18/LMS%20Technical%20Evaluation%20-%20May04.pdf. Retrieved on August 8, 2005.

OSI. Open Source Initiative, Open Source Definition, version 1.9, 2002, 2002. URL http://opensource.org/docs/definition.php. Retrieved on August 8, 2005.

OSI. Open Source Initiative, Open Source Licenses, 2005. URL http://opensource.org/licenses/. Retrieved on August 9, 2005.

C. Payne. On the Security of Open Source Software. *Information systems journal*, 12 (1):61–78, 2002.

B. Perens. Bruce Perens - Biographical Notes and Resume, 2005. URL http://perens.com/Articles/Bio.html. Retrieved on August 8, 2005.

E.S. Raymond. Goodbye, "free software"; hello, "open source", feb 1998a. URL http://www.catb.org/~esr/open-source.html. Retrieved on February 24, 2005.

E.S. Raymond. The Cathedral and the Bazaar. *First Monday*, 3(3), 1998b. URL http://www.firstmonday.org/issues/issue3_3/raymond/. Retrieved on March 30, 2005.

E.S. Raymond. Homesteading the Noosphere. *First Monday*, 3(10), 1998c. URL http://www.firstmonday.org/issues/issue3_10/raymond/index.html. Retrieved on March 30, 2005.

Sakai. About Sakai Web Page, 2005. URL http://www.sakaiproject.org/index.php?option=com_content&task=view&id=103&Itemid=208. Retrieved on August 8, 2005.

W. Scacchi. Understanding the Requirements for Developing Open Source Software Systems. In *IEEE Proceedings – Software*, volume 149, pages 24–29, 2002. URL http://www1.ics.uci.edu/wscacchi/Papers/New/Understanding-OS-Requirements.pdf. Retrieved on March 30, 2005.

D.C. Sharma. IBM tests new ways to support open source, 2005. URL http://news.com.com/2100-7344_3-5595935.html. Retrieved on August 8, 2005.

D.C. Sharma. Indian president calls for open source in defense, 2004. URL http://news.com.com/2100-7344_3-5259836.html. Retrieved on August 8, 2005.

R. Stallman. Why Software Should Be Free. URL http://www.gnu.org/philosophy/shouldbefree.html. Retrieved on August 8, 2005, On FSF homepage, 1992.

Surf. Wat heeft ....? Een overzicht van ICT&O en ELO websites van hoger onderwijs instellingen in Nederland en Vlaanderen, 2005a. URL http://e-learning.surf.nl/e-learning/watheeft#universiteiten%20nederland. Retrieved on August 8, 2005.

Surf. Surf Homepage, 2005b. URL http://www.surf.nl/en/home/index.php. Retrieved on June 7, 2005.

UniKöln. eLearning an der Universität zu Köln, 2005. URL http://www.ilias.uni-koeln.de/. Retrieved on August 8, 2005.

H.R. Varian and C.M. Varian. MOXIE: Microsoft Office-Linux Interoperability Experiment. *ACM Queue*, 1(5), July/August 2003. URL http://www.acmqueue.com/modules.php?name=Content&pa=showpage&pid=55. Retrieved on August 10, 2005.

VUB. Van Blackboard naar PointCarré; Rapport Keuze en implementatie teleleerplatform VUB, 2004. URL http://elearning.surf.nl/docs/e-learning/rapport_van_blackboard_naar_pointcarre.pdf. Retrieved on August 2, 2005.

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

D. Wheeler. Generally Recognized ad Mature (GRAM) OSS/FS programs. URL http://dwheeler.com/gram.html. Retrieved on August 11, 2005, 2004.

D. Wheeler. How to evaluate Open Source / Free Software (OSS/FS) Programs. URL http://www.dwheeler.com/oss_fs_eval.html. Retrieved on February 17, 2005, 2005.

Wikipedia. Wikipedia.org page on BSD, 2005a. URL http://en.wikipedia.org/wiki/BSD. Retrieved on January 18, 2005.

Wikipedia. Wikipedia.org page on Managed Learning Environments, 2005b. URL http://en.wikipedia.org/wiki/Managed_Learning_Environment. Retrieved on August 12, 2005.

Wikipedia. WikiPedia entry on TeX, 2005c. URL http://en.wikipedia.org/wiki/TeX. Retrieved on August 8, 2005.

H. Yamagata. The Pragmatist of Free Software: Linus Torvalds Interview. Hotwired Japan, 1997. URL http://kde.sw.com.sg/food/linus.html. Retrieved on August 8, 2005.

Appendix A

The Open Source Definition

Source: `http://www.opensource.org/docs/definition.php`, Retrieved May 11, 2005

The Open Source Definition

Version 1.9

- *The indented, italicized sections below appear as annotations to the Open Source Definition (OSD) and are not a part of the OSD. A plain version of the OSD without annotations can be found here.*

- *A printable version of this annotated page is available here.*

- *A PDF poster of the OSD is also available.*

Introduction

Open source doesn't just mean access to the source code. The distribution terms of open-source software must comply with the following criteria:

1. Free Redistribution The license shall not restrict any party from selling or giving away the software as a component of an aggregate software distribution containing programs from several different sources. The license shall not require a royalty or other fee for such sale.

- ***Rationale:*** *By constraining the license to require free redistribution, we eliminate the temptation to throw away many long-term gains in order to make a few short-term sales dollars. If we didn't do this, there would be lots of pressure for cooperators to defect.*

2. Source Code The program must include source code, and must allow distribution in source code as well as compiled form. Where some form of a product is not distributed with source code, there must be a well-publicized means of obtaining the source code for no more than a reasonable reproduction cost—preferably, downloading via the Internet without charge. The source code must be the preferred form in which a programmer would modify the program. Deliberately obfuscated source code is not allowed. Intermediate forms such as the output of a preprocessor or translator are not allowed.

- ***Rationale:*** *We require access to un-obfuscated source code because you can't evolve programs without modifying them. Since our purpose is to make evolution easy, we require that modification be made easy.*

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.

- ***Rationale:*** *The mere ability to read source isn't enough to support independent peer review and rapid evolutionary selection. For rapid evolution to happen, people need to be able to experiment with and redistribute modifications.*

4. Integrity of The Author's Source Code The license may restrict source-code from being distributed in modified form only if the license allows the distribution of 'patch files" with the source code for the purpose of modifying the program at build time. The license must explicitly permit distribution of software built from modified source code. The license may require derived works to carry a different name or version number from the original software.

- ***Rationale:*** *Encouraging lots of improvement is a good thing, but users have a right to know who is responsible for the software they are using. Authors and maintainers have reciprocal right to know what they're being asked to support and protect their reputations.*

- *Accordingly, an open-source license must guarantee that source be readily available, but may require that it be distributed as pristine base sources plus patches. In this way, "unofficial" changes can be made available but readily distinguished from the base source.*

5. No Discrimination Against Persons or Groups The license must not discriminate against any person or group of persons.

- ***Rationale:*** *In order to get the maximum benefit from the process, the maximum diversity of persons and groups should be equally eligible to contribute to open sources. Therefore we forbid any open-source license from locking anybody out of the process.*

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

6. No Discrimination Against Fields of Endeavor The license must not restrict anyone from making use of the program in a specific field of endeavor. For example, it may not restrict the program from being used in a business, or from being used for genetic research.

- ***Rationale:*** *The major intention of this clause is to prohibit license traps that prevent open source from being used commercially. We want commercial users to join our community, not feel excluded from it.*

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.

- ***Rationale:*** *This clause is intended to forbid closing up software by indirect means such as requiring a non-disclosure agreement.*

8. License Must Not Be Specific to a Product The rights attached to the program must not depend on the program's being part of a particular software distribution. If the program is extracted from that distribution and used or distributed within the terms of the program's license, all parties to whom the program is redistributed should have the same rights as those that are granted in conjunction with the original software distribution.

- ***Rationale:*** *This clause forecloses yet another class of license traps.*

9. License Must Not Restrict Other Software The license must not place restrictions on other software that is distributed along with the licensed software. For example, the license must not insist that all other programs distributed on the same medium must be open-source software.

- ***Rationale:*** *Distributors of open-source software have the right to make their own choices about their own software.*

- *Yes, the GPL is conformant with this requirement. Software linked with GPLed libraries only inherits the GPL if it forms a single work, not any software with which they are merely distributed.*

10. License Must Be Technology-Neutral No provision of the license may be predicated on any individual technology or style of interface.

- ***Rationale:*** *This provision is aimed specifically at licenses which require an explicit gesture of assent in order to establish a contract between licensor and licensee. Provisions mandating so-called "click-wrap" may conflict with important methods of software distribution such as FTP download, CD-ROM anthologies, and web mirroring; such provisions may also hinder code re-use. Conformant licenses must allow for the possibility that (a) redistribution of the software will take place over non-Web channels that do not support click-wrapping of the download, and that (b) the covered code (or re-used portions of covered code) may run in a non-GUI environment that cannot support popup dialogues.*

Appendix B

Community Activity

Please turn to the next page for the community activity examples.

Drupal

Support | Downloads | Handbook | Cont

Home » forums » Support

"How do I"-style questions

- Active forum discussions.
- Login to post a new forum topic.

Topic	Replies	Created	Last reply
How do I install and use "node_example" ?	2	3 days 5 hours ago by FranckL	2 hours 4 min ago by FranckL
Image and Image Asst both have this error	2	23 hours 43 min ago by mwoodwar	2 hours 7 min ago by mwoodwar
Making drupal available on both LAN and WAN	0	3 hours 13 min ago by Oceria	n/a
Smileys - useing urlfilter as a template	1	4 hours 48 sec ago by rcarson	3 hours 58 min ago by rcarson
firefox is faster than IE on drupal's site,why?	1	7 hours 37 min ago by robertb9527	4 hours 25 min ago by adrian
Forum Set Up Issues	0	4 hours 39 min ago by MsFreud	n/a
Possible to create sports league tables etc?	5	1 day 12 hours ago by OK Computer	6 hours 43 min ago by ramdak5000
How do I... Send a welcome message to users I create manually?	1	12 hours 36 min ago by neofactor	6 hours 46 min ago by ramdak5000
4.6 Image module (no batch uploads anymore? say it ain't so)	13	2 days 9 hours ago by zachharkey	7 hours 58 min ago by chx
Invisible content on login page	1	15 hours 22 min ago by Bure	8 hours 1 min ago by Downward Spiral
Taxonomy / Categories HELP	1	10 hours 33 min ago by richielee14	9 hours 58 min ago by Morbus Iff
eCommerce Module - How to Set Access Products Permission	3	1 week 15 hours ago by jeustace	10 hours 37 min ago by jeustace
Deleting a User	0	13 hours 24 min ago by finchwizard	n/a
Abandon taxonomy_menu for book navigation blocks...	11	2 days 20 hours ago by bomarmonk	14 hours 36 min ago by clydefrog
search forums only?	0	15 hours 24 min ago by chall3ng3r	n/a

Figure B.1: Example of an active community – Drupal 'How-To' subforum

This is the Drupal 'How-To' subforum. With over 3500 topics and over 12000 posts, the most active subforum. This subforum is meant for asking questions about using the Drupal Content Management System. The topics visible here go back less than a day, and most have multiple replies. The Drupal forum consists of three categories and a total of 19 subforums, 10437 topics and 44169 posts as of April 21, 2005. The first posts were made in June 2002, the first posts in this subforum in January 2004.

Source: `http://drupal.org/forum/22` Retrieved April 21, 2005

lucid CMS

Welcome Manual Forum Bugs/To-Do Downloads

Support this project | SOURCEFORGE.net | Take back the web

Having trouble getting up and running? Ask for help here.

Installation Support : Community Support

Goto: Forum List • New Topic • Search • Log In

Current Page: 1 of 2 **Goto Page:** 1 2 Next

Subject	Views	Posts	Started By	Last Post
Menu inactif	36	2	fmelle	04/18/05 05:30PM Last Post by zachofalltrades
Problom about installing/come here	35	3	evan	04/11/05 05:33AM Last Post by howdydoody
installation problem	113	5	mall	03/18/05 05:39AM Last Post by zachofalltrades
white page question again, sqlite edition.	205	10	rolfijn	01/26/05 11:19AM Last Post by rolfijn
another white page question	211	8	rolfijn	01/20/05 10:57AM Last Post by rolfijn
Can't get past Page 4 of install	144	3	adam	01/20/05 08:36AM Last Post by adam
Problem during install	153	3	mato	01/01/05 06:08PM Last Post by mato
SQLite Questions	116	3	Gary	12/22/04 12:59AM Last Post by zachofalltrades
DocType not pulling	79	3	Gary	12/17/04 03:38AM Last Post by Gary
BUG found ?	186	2	phaziz	12/15/04 03:56PM Last Post by Benjamin
snippets plugin Goto Page: 1, 2	153	12	ademack	11/27/04 07:10PM Last Post by zachofalltrades
upgrading b0009 to b0010	228	3	zachofalltrades	11/14/04 10:06PM Last Post by zachofalltrades
b0009 Installed locally Goto Page: 1, 2	139	20	howdydoody	11/12/04 03:38AM Last Post by wantmoore
Can't do clean install in existing DB	247	6	wantmoore	11/11/04 06:35AM Last Post by wantmoore
No access to login Goto Page: 1, 2	237	16	howdydoody	10/31/04 10:44PM Last Post by howdydoody
lucid tags in page	255	8	bobg	10/27/04 07:06AM Last Post by zachofalltrades

Figure B.2: Example of a not very active community: lucidCMS How-to support forum

This is the lucidCMS How-to support forum. The visible posts go back almost six months. Most threads have multiple replies. The forum consists of 8 subforums, 153 topics and 759 posts. The first posts were made during October 2004. This shows that this project has not been around very long, though the activity level in general is still quite low. It could have potential in time but for current evaluation of this project as a candidate, the community activity is too low.

Source:`http://forum.lucidcms.net/list.php?1` Retrieved April 21, 2005

Appendix C

Case Study – Functional Requirements

This list of functional requirements was used in the case study. This list is constructed using the main features from a website that gives detailed CMS overviews: Edutools (2005a) and a comparison report: COL (2003).

- Course based – the system is based on courses, each course has its own area in which the following features are present:
  - Basic course information – the teacher must be able to place basic information about the course (description, requirements for passing, information on the teachers, information about lectures).
  - File sharing – there is some type of file sharing available so that teachers can post files in a course area for the students.
  - Communication – the teacher has means of communicating with the students through the course interface, for example through announcements posted on the course area website and/or through an e-mail interface in the course area.
  - Groups – there is functionality to form groups of students in a course, the teacher can control the assignment of the groups. The groups can share files and communicate through the group area.
  - Tests and quizzes – There should be a possibility for the teacher to create online tests and quizzes for the students, either for self-assessment or for formal assessment. The possibility of posing a deadline, at which time a student should have finished and after which the test is closed, should be present.
- Access control – there has to be a system in place to grant various levels of access for administrators, teachers and students, each with their own rights. A link to an existing system of access for providing login information, such as a database or LDAP system, is strongly preferable.
- Administration – a system administrator must be able to easily add courses and assign teachers in the system.
- Language – the software needs to be available in English.

Appendix D

Use of CMSs in Dutch Universities

This is a list of Course Management Systems and the Dutch universities that use them, constructed using the information of the Surf[1] e-learning site on the supported systems of higher education in the Netherlands and Belgium (Surf, 2005a).

Blackboard
- Erasmus
- Katholieke Universiteit Nijmegen (KUN)
- Rijksuniversiteit Groningen (RUG)
- Technische Universiteit Delft (TUDelft)
- Universiteit Leiden – combined with TeleTop, TeleTop only at Faculty of Law
- Universiteit van Tilburg (UvT)
- Universiteit van Utrecht (UU) – certain faculties, others use WebCT
- Universiteit van Amsterdam (UvA)
- Vrije Universiteit (VU)
- Wageningen Universiteit (WAU)

WebCT
- Universiteit van Utrecht (certain faculties, others use Blackboard)

TeleTop (created by UTwente)

- Universiteit Twente (UT)
- Universiteit Leiden (Faculty of Law only, remainder is Blackboard)

StudyWeb (self-made system of TUe)

- Technische Universiteit Eindhoven (TUe)

StudieNet (self-made system of OU)

- Open Universiteit (OU) – distance education University

[1] higher education and research partnership organisation for network services and information and communications technology (ICT). (Surf, 2005b) `http://www.surf.nl`

Appendix E

Candidate List

The following table lists all the candidates investigated in the selection step of the case study. Two resources were used in constructing this list: The Commonwealth of Learning LMS Report (COL, 2003), marked in the 'C' column, and the Edutools.info website Edutools (2005a), marked in the 'E' column.

Table E.1: Case Study Candidate List

Candidate	C	E	Website
ARIADNE	x		http://www.ariadne-eu.org/
ATutor	x	x	http://www.atutor.com
Bazaar	x	x	http://ts.mivu.org
Bodington Commons	x	x	http://www.bodington.org
BSCW	x		http://bscw.fit.fraunhofer.de/
CHEF	x	x	http://www.chefproject.org
Claroline	x	x	http://www.claroline.net
Classweb	x	x	http://classweb.ucla.edu
Colloquia	x		http://www.colloquia.net
Coursemanager		x	http://www.coursemanager.com/
Coursework	x	x	http://getcoursework.stanford.edu/
COSE VLE	x		http://www.staffs.ac.uk/COSE/
Cyberprof	x		http://www.howhy.com/home/
Dokeos			http://www.dokeos.com
DotLRN	x	x	http://dotlrn.org
Eledge	x	x	http://eledge.sourceforge.net/
FLE3	x	x	http://fle3.uiah.fi
Ganeesha	x		http://www.anemalab.org/commun/english.htm
ILIAS	x	x	http://www.ilias.uni-koeln.de/ios/index-e.html
Jones E-Education		x	http://www.jonesadvisorygroup.com/index.php
Kewl	x	x	http://kewl.uwc.ac.za/
LON-CAPA	x	x	http://www.lon-capa.org
Manhattan	x	x	http://manhattan.sourceforge.net/
MimerDesk	x	x	http://www.mimerdesk.org/community/engine.html?page=2
Moodle	x	x	http://www.moodle.org
OpenCourse	x		http://www.opencourse.net

Candidate	**C**	**E**	**Website**
OCW open courseware	x		http://ocw.mit.edu
OLMS	x		http://www.psych.utah.edu/learn/olms/
Opaltree	x		http://www.opaltree.com/
OpenLMS	x		http://openlms.sourceforge.net
OpenUSS	x		http://openuss.sourceforge.net
Ripples/Manic	x		http://ripples.cs.umass.edu/
Sakai		x	http://www.sakaiproject.org/
Shadow netWorkSpace	x		http://sns.internetschools.org/~ischools/info/sns2/index.cgi
Uni Open Platform	x		http://uni-open-platform.fernuni-hagen.de/
WhiteBoard	x	x	http://whiteboard.sourceforge.net

Appendix F

Selection Results

F.1 Step 1: Elimination

In the first step of the selection process a number of candidates was eliminated that did not meet certain criteria. The values of these criteria and remarks are shown in the table below.

Table F.1: Selection by Elimination

Project	Last release	FR	Remarks
ARIADNE	?	No	Does not appear to be a CMS but some e-learning related tools.
ATutor	2005–04–02	Yes	
Bazaar	2004–04–08	Yes	
Bodington Commons	2005–11–05	Yes	
BSCW	2004–12–11	No	Not a CMS but a group collaboration tool.
CHEF	N/A	N/A	Is discontinued, efforts have been moved to Sakai project.
Claroline	2005–26–04	Yes	Project has split with some of the developers, Dokeos is a spinoff.
Classweb	2002–27–09	?	Website will not load. web.archive.org shows links to SourceForge and FreshMeat. release dates are from there.
Colloquia	2002–30–10	Yes	Very group based, possibly not really a CMS.
Coursemanager	?	Yes	Can not find download or evidence that this is Open Source.
Coursework	N/A	N/A	Continued with Sakai project.
COSE VLE	?	Yes	Need to register to download. Very little information.

Project	Last release	FR	Remarks
Cyberprof	?	?	Announcement of upcoming release on July 1, 2001! No download available.
Dokeos	2004–21–09	Yes	Spinoff of Claroline.
DotLRN	2005–12–01	Yes	
Eledge	2003–21–10	Yes	Website info is minimal.
FLE3	2005–01–04	Yes	
Ganeesha	2005–25–01	No	Full CMS but only in French at the moment. A multilingual version may be released in the future. For the moment, only a summary page with minimal information is available in English.
ILIAS	2005–22–04	Yes	German based, and it shows.
Jones E-Education	2004–05–04	Yes	License is not really Open Source.
Kewl	?	?	The Kewl page appears to be an installation. Links to downloads give no additional information and require registration for downloading. Found information on KEWL.nextgen but states on the site it is still in very early stages. Not currently a viable candidate.
LON-CAPA	2005–26–03	Yes (?)	Appears to have much work in progress.
Manhattan	2004–13–05	Yes	
MimerDesk	2003–15–11	Yes	Not sure if it is course based.
Moodle	2005–07–05	Yes	
OpenCourse	2002–21–09	No	Beta, apparently no development anymore.
OCW open courseware	N/A	No	Not a CMS, MIT educational resource of MIT course materials.
OLMS	2004–27–02	?	Almost no information on the website.
Opaltree	N/A	N/A	No release yet, according to COL LMS report had a target release of august 2003. Press release and download page say 'Under construction'.
OpenLMS	?	Yes	Can only find a demo download dated Feb 2003.
OpenUSS	2004–23–06	Yes	

Project	Last release	FR	Remarks
Ripples/Manic	?	?	Can not find download or any info on the software itself.
Sakai	2005–07–03	Yes	Very young project. Check the existing features further.
Shadow netWorkSpace	2002–25–10	Yes	
Uni Open Platform	2004–10–09	Yes	
WhiteBoard	2003–07–08	Yes	

F.2 Functionality Scores

Table F.2: Functionality Scores

Project	Score	Remarks
ATutor	10	ATutor's main priority is accessibility, holding to standards of W3C for website accessibility. The feature set seems complete.
Bazaar	6	No feature description on the website. In the demo, the features seem very basic.
Bodington Commons	4	Little information on features. No online demo.
Claroline	6	Online demo. Features seem complete. It seems like they have used Blackboard as a template. Components are reused, such as forum which is built from phpBB.
COSE VLE	2	No demo, feature list is small.
Dokeos	6	Demo, screenshots. Feature set seems complete.
DotLRN	6	Feature list on website seems complete, no demo.
Eledge	2	Demo does not work. Login page looks very basic. No feature list on the website.
FLE3	4	Demo and screenshots. Interface is a bit odd. Features seem somewhat basic.
ILIAS	8	Features seem complete though the interface takes quite some getting used to.
LON-CAPA	6	Features seem reasonably complete though some features are explicitly excluded that might be important, and the interface seems very basic.
Manhattan	4	No demo, features list seems reasonable.
MimerDesk	6	Feature list and demo. Seems complete.
Moodle	10	The Moodle website is built in Moodle itself. There are a few demo courses that show many different features. Feature set is very complete.

Project	Score	Remarks
OLMS	2	Demo is very basic, only see announcements, not sure if any other are features present.
OpenUSS	4	Basic features seem present according to website info, no demo.
Sakai	6	Not sure if feature set is complete. Young project.
Uni Open Platform	2	Feature list is small. Online demo shows absolute minimal functionality. Not complete.
WhiteBoard	2	Is built after Blackboard, features seem to be basically implemented.

F.3 Community Scores

Table F.3: Community Scores

Project	Score	Topics	Posts	Remarks
ATutor	6	1140	4247	Support forums, since May 10 2003.
Bazaar	2			Mailing list. No information on the list, need to subscribe to get in.
Bodington Commons	4		184	SourceForge mailing list development since April. Recent move to this list?
Claroline	8	3186	10998	Forums on use, development, multilingual and bugs, since March 2002.
COSE VLE	1			No visible signs of a community. Just a guestbook.
Dokeos	8	3226	12882	Community section on mainpage lists forum, wiki, users, customers, team. The latter three gives a list of each (with option to add to users). Forums since March 2002 (Posts overlap with Claroline!).
DotLRN	5	990		.LRN Q&A forum on OpenACS forums, 990 threads, no info on number of posts, since June 2002.
Eledge	4		277	Forums on SourceForge but no link there from the mainpage. Since Dec 2001.

Project	Score	Topics	Posts	Remarks
FLE3	3			Mailing lists (announce, users, dev) with archives. Activity is low (No posts at all in users March, February, November, the rest is about 5 posts a month. No totals, dev a little more, sometimes 10 posts a month, but some months nothing).
ILIAS	8	2505	14422	Forums, subforums spread thin, with some < 10 topics a piece.
LON-CAPA	4			Mailing list with archives, between 10 and 80 posts a month.
Manhattan	3			SourceForge mailing list with archives, about 10 posts a month.
MimerDesk	2			Need to login to get to the main 'community', but inside cannot find MimerDesk group.
Moodle	10	14367		Moodle community announcement on mainpage. Links to relevant forums everywhere. Moodle website is built in Moodle itself. Community is very active. The main 'course', 'Using Moodle' has 14367 topics and there are seperate courses for each language Moodle is translated in and several other courses such as documentation and business uses.
OLMS	2		3	Three messages in SourceForge forum since early 2004, no mailing lists.
OpenUSS	2		12	Site appears to be built in OpenUSS, hard to navigate to get to any discussion. Discussion is dated (2002), mailing list has some more recent posts but only 1 in 2005, 4 in 2004, 3 in 2003.
Sakai	5		1399	Site is built in Sakai. No figures on number of posts, estimated – 110 on user + developer discussion, e-mail acrhives showing 865 messages in development, 424 in users.

Project	Score	Topics	Posts	Remarks
Uni Open Platform	1			Registration leads to a site about this type of software, cannot find any discussion related specifically to Uni Open Platform.
WhiteBoard	2		6	Forum on SourceForge.

F.4 Release Activity Scores

Table F.4: Release Activity Scores

Project	Score	Last Release	2004/5	Wgt	Remarks
ATutor	8	2005–04–02	6	8	
Bazaar	2	2004–04–08	3	2	
Bodington C.	6	2005–12–05	9	3	
Claroline	6	2005–26–04	7	5	
COSE VLE	0	?	?	?	
Dokeos	4	2004–21–09	3	6	Dokeos forked from Claroline 2004.
DotLRN	4	2005–12–01	4	4	
Eledge	1	2003–21–10	0	0	
FLE3	3	2005–01–04	1	4	1.4.5 in nov 2003, 1.5 in apr 2005.
ILIAS	6	2005–22–04	7	4	ILIAS has 2 major versions (ILIAS2 and 3) under development. Apart from stable releases some beta releases were made.
LON-CAPA	5	2005–25–03	13	2	Releases estimated. FreshMeat info goes back to July 2004.
Manhattan	2	2004–13–05	3	3	
MimerDesk	1	2003–15–11	0	0	
Moodle	10	2005–07–05	13	5	
OLMS	1	2004–27–02	1	4	
OpenUSS	2	2004–23–06	1	3	All very unclear.
Sakai	3	2005–07–03	2	5	Project's first release in Oct. 2004.
Uni O.P.	2	2004–10–09	?	?	Very little information on releases.
WhiteBoard	0	2003–07–08	0	0	

F.5 Longevity Scores

Table F.5: Longevity Scores

Project	Score	1st Release	Remarks
ATutor	7	01/11/2002	Mentions in various articles on e-learning systems, recommended by COL LMS report.
Bazaar	10	09/23/2000	
Bodington Commons	4	08/15/2003	Date is from release 2.10 RC1, cannot find data on earlier releases.
Claroline	7	03/01/2002	Article in french Linux magazine. Date is from first forum posts.
COSE VLE	0	?	No information on releases.
Dokeos	2	04/23/2004	Date is first release after separation from Claroline, the software is the same age as Claroline. Was chosen as primary CMS for Brussels university.
DotLRN	5	03/30/2003	Shortlisted in COL LMS report.
Eledge	7	12/27/2001	
FLE3	7	02/15/2002	
ILIAS	10	09/25/2000	Recommended in COL LMS report.
LON-CAPA	7	02/01/2002	From mailing list archives.
Manhattan	9	01/30/2001	
MimerDesk	7	01/10/2002	
Moodle	6	08/20/2002	Shortlisted in COL LMS report, article in Linux magazine, chosen to be primary CMS for Dublin University.
OLMS	3	02/27/2004	
OpenUSS	9	04/05/2001	
Sakai	1	10/18/2004	
Uni Open Platform	6	06/14/2002	Release of 2.0, cannot find older releases.
WhiteBoard	5	11/04/2002	

F.6 Total Selection Scores

The table with the total scores is shown below.

The scores of each criterion are shown in the first columns (F = Functionality, C = Community, R = Release Activity, L = Longevity) and the calculated Total scores in the last column. These are calculated using the weights from table F.7.

Table F.6: Total Selection Scores

Project	F	C	R	L	Total
Moodle	10	10	10	6	10
ATutor	10	6	8	7	8
ILIAS	8	8	6	10	8
Claroline	6	8	6	7	7
Dokeos	6	8	4	2	6
LON-CAPA	6	4	5	7	5
DotLRN	6	5	4	5	5
Bodington Commons	4	4	6	4	5
Sakai	6	5	3	1	4
Bazaar	6	2	2	10	4
FLE3	4	3	3	7	4
Manhattan	4	3	2	9	4
MimerDesk	6	2	1	7	4
OpenUSS	4	2	2	9	3
Eledge	2	4	1	7	3
Uni Open Platform	2	1	2	6	2
OLMS	2	2	1	3	2
WhiteBoard	2	2	0	5	2
COSE VLE	2	1	0	0	1

Table F.7: Weights

Criterion	Weight
Functionality	35
Community	30
Release Activity	25
Age	10
Total	100

F.7 Chart

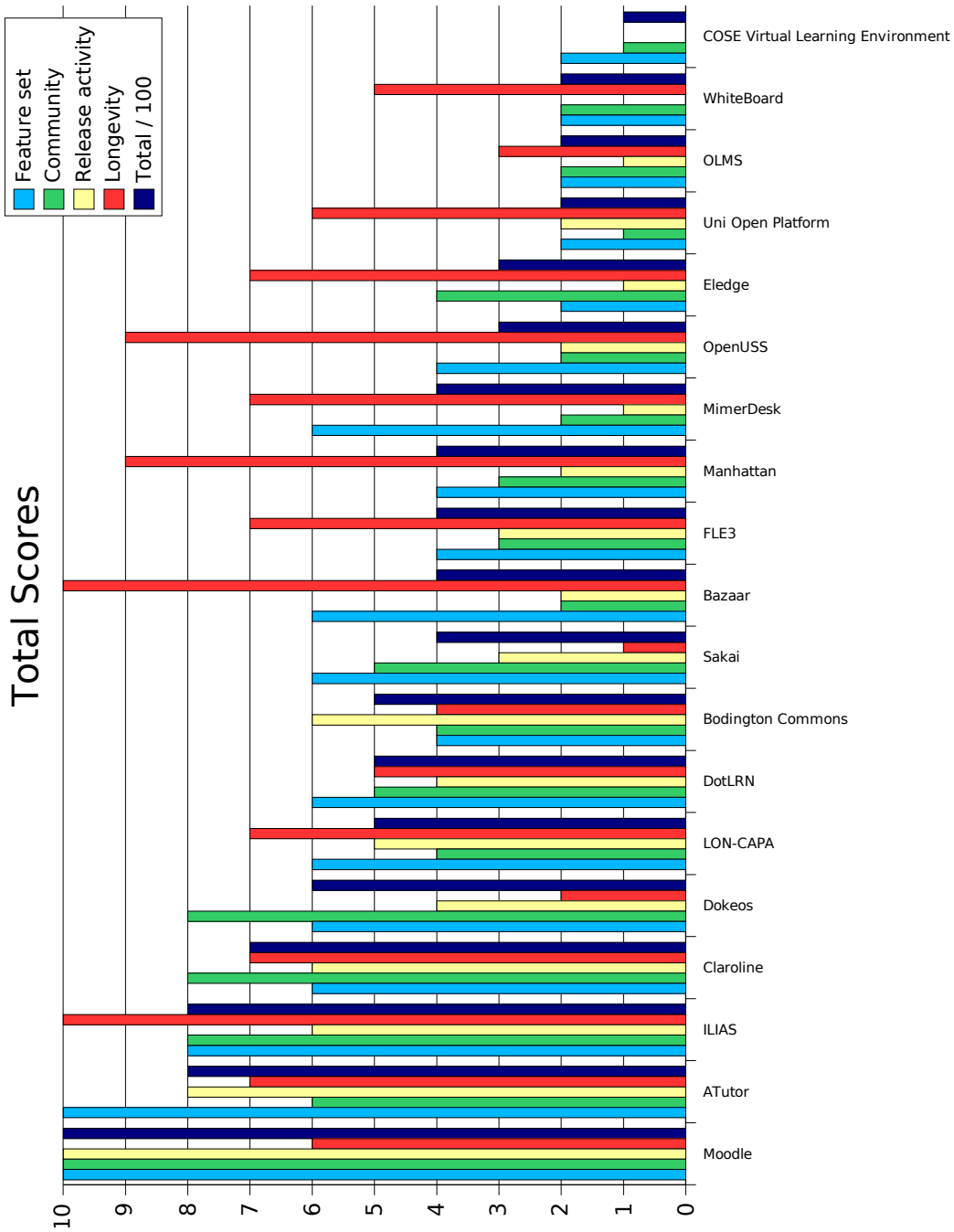

Figure F.1: Chart of the Selection Scores

Appendix G

Moodle Philosophy

Source: `http://moodle.org/doc/?frame=philosophy.html`

G.1 Philosophy

The design and development of Moodle is guided by a particular philosophy of learning, a way of thinking that you may see referred to in shorthand as a 'social constructionist pedagogy'. (Some of you scientists may already be thinking 'soft education mumbo jumbo' and reaching for your mouse, but please read on – this is useful for every subject area!)

This page tries to explain in simple terms what that phrase means by unpacking four main concepts behind it. Note that each of these is summarising one view of an immense amount of diverse research so these definitions may seem thin if you have read about these before.

If these concepts are completely new to you then it is likely that these ideas will be hard to understand at first – all I can recommend is that you read it carefully, while thinking about your own experiences of trying to learn something.

G.1.1 Constructivism

This point of view maintains that people actively construct new knowledge as they interact with their environment.

Everything you read, see, hear, feel, and touch is tested against your prior knowledge and if it is viable within your mental world, may form new knowledge you carry with you. Knowledge is strengthened if you can use it successfully in your wider environment. You are not just a memory bank passively absorbing information, nor can knowledge be 'transmitted' to you just by reading something or listening to someone.

This is not to say you can't learn anything from reading a web page or watching a lecture, obviously you can, it's just pointing out that there is more interpretation going on than a transfer of information from one brain to another.

G.1.2 Constructionism

Constructionism asserts that learning is particularly effective when constructing something for others to experience. This can be anything from a spoken sentence or an internet posting, to more complex artifacts like a painting, a house or a software package.

For example, you might read this page several times and still forget it by tomorrow – but if you were to try and explain these ideas to someone else in your own words, or produce a slideshow that explained these concepts, then I can guarantee you'd have a better understanding that is more integrated into your own ideas. This is why people take notes during lectures, even if they never read the notes again.

G.1.3 Social Constructivism

This extends the above ideas into a social group constructing things for one another, collaboratively creating a small culture of shared artifacts with shared meanings. When one is immersed within a culture like this, one is learning all the time about how to be a part of that culture, on many levels.

A very simple example is an object like a cup. The object can be used for many things, but its shape does suggest some 'knowledge' about carrying liquids. A more complex example is an online course – not only do the 'shapes' of the software tools indicate certain things about the way online courses should work, but the activities and texts produced within the group as a whole will help shape how each person behaves within that group.

G.1.4 Connected and Separate

This idea looks deeper into the motivations of individuals within a discussion. '**Separate**' behaviour is when someone tries to remain 'objective' and 'factual', and tends to defend their own ideas using logic to find holes in their opponent's ideas. '**Connected**' behaviour is a more empathic approach that accepts subjectivity, trying to listen and ask questions in an effort to understand the other point of view. '**Constructed**' behaviour is when a person is sensitive to both of these approaches and is able to choose either of them as appropriate to the current situation.

In general, a healthy amount of connected behaviour within a learning community is a very powerful stimulant for learning, not only bringing people closer together but promoting deeper reflection and re-examination of their existing beliefs.

G.1.5 Conclusion

Once you are thinking about all these issues, it helps you to focus on the experiences that would be best for learning from the learner's point of view, rather than just publishing and assessing the information you think they need to know. It can also help you realise how each participant in a course can be a teacher as well as a learner. Your job as a 'teacher' can change from being 'the source of knowledge' to being an influencer and role model of class culture, connecting with students in a personal way

that addresses their own learning needs, and moderating discussions and activities in a way that collectively leads students towards the learning goals of the class.

Obviously Moodle doesn't force this style of behaviour, but this is what it is best at supporting. In future, as the technical infrastructure of Moodle stabilises, further improvements in pedagogical support will be a major direction for Moodle development.

Appendix H

ATutor Philosophy

Source: `http://atutor.ca/philosophy.php` Retrieved July 26, 2005.

H.1 Introduction

In designing ATutor we have had the specific goal in mind of creating an adaptive learning environment that anyone could use. Regardless of how people go about learning, and regardless of the technology they might be using to learn online, ATutor is designed to accommodate all learners.

Human learning is highly complex, and ATutor can't hope to adapt to all the intricacies in the ways people interact with the world (anytime soon `;-)`). A simplified six point model that draws on popular understanding of learning and the structure of knowledge, provides a starting point for developing an intelligent learning environment that adapts to all who use it.

H.2 Learning and Knowledge

Underlying ATutor's navigation structures and visual presentation is an understanding of the perceptual forms information takes on, an understanding of the senses through which people prefer to absorb and process information, and an understanding of the structural representations knowledge takes on in memory.

H.2.1 Perceptual Learning and Processing Styles

Perceptual style refers to the tendency people have to lean toward learning strategies and learning situations that favour their visual, verbal, or their kinesthetic faculties. These faculties roughly correspond to abilities of imagery, auditory/verbal processing, and physical coordination. In most cases learners use all three faculties, but tend to prefer one over the others.

1. Visual

   Visual learners like to see or imagine things as their preferred means of learning. They learn by watching or viewing information. Knowledge tends to be

represented in pictures. Architects, artists, and engineers tend to be visual learners.

2. Verbal

   Verbal learners like to hear or verbalize things as their preferred means of learning. They learn through hearing, saying and reading information. Knowledge takes on an auditory nature as new information is being absorbed. Writers, speakers, and public personalities tend to be verbal learners.

3. Kinesthetic

   Kinesthetic learners like to do or experience things as their preferred means of learning. They learn through activities and movement. Knowledge takes on a physical feeling of the circumstances under which learning occurred. Athletes, inventors, and craftsmen tend to be kinesthetic learners.

H.2.2 Structural Representations of Knowledge

Knowledge is 'encoded' in memory in structural representations of relationships between facts and ideas. The perceptual 'sense' of information as described above, is interconnected in Webs, hierarchies, and chains that arrange knowledge into 'schemas' or 'scripts' representing units of factual information or mental procedures, respectively.

4. Global

   Global learners structure information in webs. Information is interconnected, with related topics linked to each other through weighted threads. Knowledge takes on a 'big picture' structure through understanding the general concepts within a content area, creating a framework to which more detailed information can be connected.

5. Hierarchical

   Hierarchical learners structure information in trees. More general ideas have subordinate ideas associated with them, which in turn have subordinate ideas associated with them. Knowledge takes on a structure much like a computer directory tree, with folders, sub folders, and files at varying depths.

6. Sequential

   Sequential learners structure information in chains. Topics begin and end, and follow a straight path through a sequence of ideas. Knowledge takes on a linear structure much like a time line, or step by step procedures.

H.3 Web Accessibility

Web accessibility generally refers to the 'inclusiveness' of Web content, or the ability of people with disabilities to gain access to that information using assistive technology. It can also refer to access for those using older technology to access the web, those accessing at lower bandwidths, or those accessing with limited experience or

other special needs. ATutor adopts many strategies to ensure accessibility to both learners of diverse skill and ability, and to learners using older or specialized technologies who are learning online.

H.3.1 Web Content Accessibility

The Web Content Accessibility Guidelines 1.0 published by the W3C provides the initial model of accessibility for ATutor. Guidelines represent technical issues that must be addressed to ensure that all attempting to access information can do so with relative ease. WCAG 1.0 provides the framework for creating an application that will work with any technology accessing it over the Web.

Looking ahead to WCAG 2.0, usability is addressed with greater emphasis, providing guidelines for accommodating abilities as well as technologies. Web content adaptability addresses the inclusive usability of information.

H.3.2 Web Content Adaptability

Based on the six point model outlined above, ATutor presents information in visual, verbal, and kinesthetic perceptual forms, as well as global, hierarchical, and sequential structural forms.

Perceptual Adaptability

1. Visual

   Icons are used throughout ATutor to represent tools, ideas, and resources in visual form. The visual layout can be adapted to each learner's preference and saved for future use, providing a consistent arrangement of ATutor features and navigation tools. Hierarchical presentations within the menus and the Sitemap also provide visual representations of the content structure. Within the Global and Local menus a learner's position within ATutor content is always highlighted, giving them a visual cue to their location within a collection of ideas.

2. Verbal

   ATutor is presented in verbal form by default. ATutor can be reduced to a text presentation alone if learners prefer to read (or listen to) content rather than view or visualize it. A text only presentation also ensures that all information can be accessed by any technology that reads HTML. ATutor development efforts also include the creation of a verbal feedback module (ATalker) that renders ATutor in audio form so learners can hear the environment as they navigate through it, and listen to content using an onthefly Text-to-Speech server.

3. Kinesthetic

   ATutor is highly interactive and consistently presented throughout. Learners 'use' ATutor to present content in a form that suits their perceptual styles, and can structure information into global webs, hierarchical trees, and sequential chains. Consistent layouts allow users to develop keyboard access strategies

creating mental sequences of physical movements, or physical procedures, allowing them to automate their use of the environment and devote more mental resources to learning the content being presented.

Structural Adaptability

4. Global

   Information can be presented in 'wholes' that allow learners to develop 'big pictures' of topic areas, familiarizing themselves with the main ideas within a larger topic as a framework for learning the finer details. The Sitemap presents an entire ATutor course as a tree of linked page titles, allowing learners to see the course topics in their entirety and jump around from topic to topic as they become relevant to ongoing learning. The Global Menu also presents the course as a whole, though the portion of the course displayed can be controlled by learners, giving them the ability to limit the amount of information presented at any one time. A course search engine also allows learners to move through content in a global manner.

5. Hierarchical

   The Sitemap and Global menu, as well as the Local menu, the breadcrumb string, heading navigation, and Table of Contents navigation, provide learners with hierarchical strategies for moving 'up and down' through ATutor content.

6. Sequential

   Next and previous links allow learners to move through ATutor content in a predefined order. If they leave the sequence of topics, to go to the discussion forums to post a message for example, they can use the resume link, or highlighted titles in the menus to return to the position in the content where they left off.

H.4 Intelligent Learning Environments

To reach our goal of creating an adaptive learning environment, navigation patterns and preference settings provide the data for tracking how learners use ATutor. In the early versions of ATutor this data is collected in a static database (see: My Tracker, or Course Tracker within ATutor), and is controlled by individual learners. In later versions this data will be under the control of the system itself, monitoring learners' click paths and modifications to their preference settings, and using the data to adapt the learning environment to match the learning tendencies of each user.

Appendix I

CMS Evaluation

I.1 Community

I.1.1 Moodle

Moodle's website, built in Moodle itself, is centered around the community. In every section there are links to relevant forums and requests for participation.

From the main page of Moodle:

'*Moodle has a large and diverse user community with over 50,000 users registered on this site alone, speaking 60 languages in 115 countries. The best place to start is 'Using Moodle', which is where the main international discussions are held in English, but we have a variety of groups discussing other topics and in other languages.*'

The documentation page, for example, links to several documentation forums, among which the 'Suggestions concerning basic Moodle documentation' forum.

As stated in the selection step, the main course on the Moodle site, 'Using Moodle', lists over 14000 topics. Other than that, there are 20 courses concerning language-specific issues, among which are Chinese, Japanese, Arab, and the major European languages. Each of these groups is concerned with the translation of Moodle, among other things. They update the language files with each update of Moodle.

The number of users in the 'Using Moodle' course is over 3000. Not all users that have signed up may be active participants though.

The bugtracker gives insight in the number of 'official' developers and their share in the project. The main developer is clearly Martin Dougiamas, who takes on 63% of the bugs. He is also a very active participant in the forums. A total of 34 developers is listed who have had bugs assigned to them, meaning that they are responsible for dealing with the issue. Twenty percent of the developers (the top seven) deals with ninety percent of the bugs.

Responses

A sample of 200 topics from the 'Using Moodle – General problems' forum was used. There were a total of 662 replies, 48 topics (24%) had no reply, 47 percent had 2 to 5

replies. The dates of the last post of these 200 topics went from May 21, 2005 to June 13, 2005. This averages to 28 posts per day.

To make an accurate comparison with projects with a different activity rate, a two month sample was used. This constitutes 516 topics, with a total of 1630 replies (average 3.16), 116 topics with 0 replies (22.5%), and 44% with 2–5 replies. An average of 26 posts a day were made. These figures do not differ significantly from the results gotten from the 200 topic sample.

The number of topics without replies is rather high. Some of these posts could include 'nonsense' posts that nobody bothered to reply to. However, a percentage this high suggests that some genuine questions are not being answered.

I.1.2 ATutor

The ATutor forums are significantly smaller, with 1140 topics and 4247 posts. The forums are being linked to as 'Support Forums' on the main page. There is also a Community Discussion section on the main page linking to the sub forum. This section was added recently (September 2004) and is meant for sharing experiences with ATutor.

The number of registered users on the ATutor website was not listed.

The ATutor project does not seem to have a public bugtracker. The development documentation states that bugs are to be reported in the bug report forum.

Responses

A sample of 200 topics from the ATutor support forum showed 611 replies, 20 (10%) topics with no replies, and the largest percentage (40%) was the topics with one reply. The period of these topics ranges from January 6, 2005 to June 13, 2005. Thus the average number of posts per day is four.

The two month sample for ATutor had 73 topics, showing a few differences: 15% with 0 replies, 41% with one reply and an average number of posts per day of three.

I.1.3 Compared

The number of responses to the 200 topic sample is about the same. Moodle's number of topics with no replies is rather high, but the activity level over time is much higher than ATutor's, both in the 200 topic sample and the two month sample. Aside from that, keep in mind that Moodle has a wide range of forums beyond the general problems forum (i.e. installation problems, and separate forums for each module in Moodle). A more detailed comparison including all forums may give different figures. It is very possible that the more detailed forums, for example for each of the modules, get more responses per topic because the people responsible for that module check that forum more often than the general forum is being checked. Further more, it is well known that more detailed questions usually get better and more answers than more general questions, which are more likely to be placed in the general forum in case of Moodle.

I.1.4 Score

Moodle: 9

ATutor: 5

I.2 Release Activity

I.2.1 Moodle

Moodle has made 26 releases since 1.0.

As seen in the selection, Moodle had made 13 of those releases since January 1 2004. When viewing the release notes to see the significance of each release, Moodle's release notes are quite long.

I.2.2 ATutor

ATutor has made a total of 14 releases. ATutor has made six releases since January 1, 2004. ATutor's release notes are shorter that Moodle's. Though some of the difference may be explained by different wording and mentioning of more detailed changes, in this case it appears that the description of the changes are reasonably equal.

I.2.3 Score

Moodle: 10

ATutor: 6

I.3 Longevity

I.3.1 Age and version

Moodle

Moodle's first version, 1.0, was released on August 20, 2002. It is now at version 1.5.

I.3.2 ATutor

ATutor's first release was version 0.9.6, on January 11, 2002. Its 1.0 release was on December 2, 2002. It is now at version 1.4.3 with 1.5 in beta.

There is no significant difference between these figures.

Moodle: 10

ATutor: 10

I.3.3 New Technology: PHP 5

Both products are built using the web language PHP. The PHP project has recently released a new main version of the language, PHP 5. This new version has not been adopted by everyone yet, but it is to be expected that it will become more and more used over the next year. There have been some issues with backwards compatibility, meaning that programs written for the previous version (PHP 4) do not always work properly in the new version, because certain functionality has changed or been deprecated.

According to the documentation, Moodle using PHP 5 is supported as of Moodle 1.4. This version was released August 31, 2004, one and a half months after the first official release of PHP 5 (Moodle, 2005e).

Though ATutor does not list PHP 5 as supported in the requirements section of the documentation (it says PHP 4.2.0 or higher, which does imply PHP 5 as well) (ATutor, 2005a), a forum post[1] does confirm that ATutor supports PHP 5. Information on the first version that was supported was not found. The ACollab tool, which can be added to ATutor to add group functionality, does not currently support PHP 5 and will probably not support it officially until the fall of 2005[2].

Moodle: 10

ATutor: 8

I.3.4 Score

Moodle: 10

ATutor: 9

I.4 License

Both ATutor and Moodle are licensed under the latest version of the GNU GPL (Version 2, June 1991). As mentioned before this is one of the most used licenses in the Open Source world.

I.4.1 Score

Moodle: 10

ATutor: 10

I.5 Support

I.5.1 Community

The level of community support depends largely on the activity level of the community in the discussion areas. This was established in the community criterion, as

[1] `http://www.atutor.ca/view/2/2469/1.html`

[2] See `http://www.atutor.ca/view/9/4567/1.html`

stated above. In order to evaluate the quality of the replies and speed of replies, in terms of being to the point and complete in answering the question, as well as how fast the first good answer was given to a question, a number of posts, with different reply numbers were chosen.

Moodle

The activity level at Moodle's community is high. This increases the chance of getting questions answered. There are a number of regular posters that often answer questions. This number is between five and ten for the general areas. With this number of people contributing to community support on a very regular basis, the chance of getting an answer is reasonably high.

The speed of replies is pretty high, many questions had a reply within an hour, almost all within a day. The fact that it takes almost a day sometimes could be due to the fact that Moodle offers a subscription service to the forums, customisable for each forum, that allows the user to either receive an email of each forum post or to receive daily digest e-mails, that give an overview of all posts made in the last 24 hours, sent out at a fixed time every day.

The quality of community support in terms of forum replies is high. Most people are very willing to help in answering questions, asking for additional information when needed. Sometimes people asking a question express themselves very poorly, showing a language barrier, but those posts are handled gracefully in most cases as well. When something asked for is not possible, often it is either already being thought about and/or being planned to be implemented, or it is seriously discussed as an option to implement it. Users also offer tutorials and workarounds that they have that are relevant to the problem. The Moodle Documentation course's forums also have a 'how-to' forum where users can contribute 'how-to' tutorials, such as 'How to put courses in the middle of the front page' and 'How to add a new activity'. This forum is pretty active, showing how willing the users are to participate and contribute to the project.

As far as the organisation goes, the areas are well defined and pretty logical considering Moodle's modular structure – each module has their own forum, and all have a good amount of activity, with multiple posts per day. Other areas include documentation and development, each with several forums.

ATutor

The activity level at ATutor's community is significantly lower than Moodle's. The chance of getting a question answered in this light is lower. One person (with user name 'greg') dominated most of the replies to support questions. This would mean a too heavy reliance on one person to get answers from, which poses a risk to community support levels. The response time varies, sometimes two hours or less, sometimes half a day, and in some cases it takes several days before a reply is posted.

The quality of the replies is somewhat lower at ATutor than observed at Moodle's community. Most issues dealt with are rather low level, giving simple solutions to problems. This is in part due to the simple questions being asked on these forums.

Not much discussion about the functionality and possibilities of the software seems to be taking place here.

The support forums are organised by the components that are available from the ATutor project, namely ATutor, ACollab (adding group functionality to ATutor), AChat and AForm. The activity in the other component's areas is quite low, but the distinction seems logical.

Moodle: 10

ATutor: 5

I.5.2 Paid Support

Moodle

Moodle's creator, Martin Dougiamas, has also founded moodle.com:

'*The Moodle Partners are a group of service companies guided by the core developers of Moodle. We provide a range of optional commercial services for Moodle users, including fully-serviced Moodle hosting, remote support contracts, custom code development and consulting.*'

(moodle.com, 2005)

Various partners are listed on this site as paid support options, from various countries, including USA, Canada, United Kingdom, Italy and Spain.

ATutor

ATutor has its own support contracts and also lists other service providers, though only one of those has support listed explicitly, the others are mostly offering hosting and custom development.

Moodle: 10

ATutor: 7

I.5.3 Score

Moodle: 10

ATutor: 6

I.6 Documentation

I.6.1 Moodle

Moodle's Documentation section is a separate course, linked from the main page's menu. It contains the following main sections:

- **About Moodle** – Information on Moodle itself, its background, philosophy, etc. and sections on the license, features, future plans, and credits.

- **Administration** – Installation instructions and FAQ[3], information on installing Apache, PHP and MySQL (the web server, programming language and database), and a section on upgrading.
- **Using Moodle** – Teacher Manual.
- **Development** – Developer's Manual, coding guide, using CVS[4] and Translation.

There is a section containing user contributed documentation (Moodle, 2005d): A student manual, an extensive teacher manual (127 pages), another manual for teachers, trainers and administrators, counting 58 pages, and there is also a link to a 240 page book that has just been published in July 2005 by O'Reilly publishers, called 'Using Moodle' (OReilly, 2005). There's also a section on presentations that users have contributed that they have used in various situations to present and promote Moodle, and a section of how-tos that are mostly user-contributed. The course also links to the appropriate forums to post questions and suggestions on the documentation. The forums are used for discussion about the documentation.

The default Moodle installation provides question mark signs with every section or option that has a section in the included online help. This help file is also included in all the translations of the software. The standard documentation is also available in a local installation by going the doc/ directory in the browser.

A few user-contributed translations of the documentation are available: Dutch, French and Italian versions of the Teacher Manual and a Spanish version of the Study Guide and Example How-To's.

The Developer documentation explains how Moodle is organised, and what the general philosophy is. It gives information on the structure of the modules, and instructions on how to add a module. It also gives code guidelines. The source code files each have a brief description of the file's purpose on top and some comments throughout the file.

I.6.2 ATutor

ATutor has a documentation section clearly marked in the menu on the main website. It contains installation instructions for three versions of the software, information on the software requirements, the configuration variables, information on the accessibility options and a FAQ section. It also gives information on how to go about creating a translation of the software and how to contribute this back to the project. There is also a 'How-To Course', which apparently contains the official documentation, according to the description on the website:

'The ATutor How-To course is the official ATutor documentation. The course describes how to use ATutor as a student, how to create and manage online courses, and how to administer an ATutor course server.'

[3] Frequently Asked Questions

[4] Concurrent Version System, used when multiple developers work on the same code, see `http://www.cvshome.org/`

This How-To course is browseable on the ATutor demo server and can also be downloaded. This is a plus, because the user documentation can be offered on the installation for the users, and since it is built in ATutor it makes use of ATutor's accessibility options and standards, so the user documentation is just as accessible as any other course content. This documentation course appears well organised and reasonably complete.

There is one translated documentation file available in Chinese.

The documentation section also links to developer documentation, which is a document containing code conventions, information on the configuration used, getting ATutor from SVN[5] and the structure of ATutor's files and databases.

A few code files, including classes, were checked for code comments. The files all had a copyright declaration at the top but no description of the purpose of the file. There were a few comments in each file to explain some sections but the number of comments was rather low. The developer documentation gives a good example of how to describe a function's use and parameters, but this type of comment in the files themselves was not found.

A discussion area or group concerning documentation could not be found. The documentation files in the How-To course were updated in September 2004.

I.6.3 Score

Moodle: 9

ATutor: 7

I.7 Security

Two of the large security advisories, SecurityFocus and Secunia, were used to search for vulnerabilities in the ATutor and Moodle software. Moodle had a few listings. ATutor had 1 vulnerability reported in both, which is very recent (June 16), and has no fix as of yet (July 11). Evidence that the ATutor team is even aware of this vulnerability at the moment was not found.

The advisories showed six to seven vulnerabilities for Moodle. Both had listed mostly the same alerts. All vulnerabilities are patched in the latest version. The vulnerabilities were reported by the vendor in almost all cases, meaning the the Moodle team reported the problem themselves after providing a new version of the software that solves it, so the users can be alerted to the problem so they can upgrade. A vulnerability report containing a number of problems in the bugtracker was responded to almost immediately (within one hour) and the stable release containing the fixes was released a week after the problem was reported in the bugtracker.

Moodle has a public bugtracker in which any problems are reported and the status of new features is being kept. The reaction time to critical problems in the core of Moodle is very good, in less than a day most problems are solved. The modules' problems are assigned to the modules' creators and sometimes take a little longer, something

[5]Subversion, a version management system similar to CVS. See `http://subversion.tigris.org/`

to be expected with add-ons like this. Still, time between problem discovery and fix is less than a week.

ATutor does not have a public bugtracker, just a forum to report problems. The level of activity there is not very high. The reaction time cannot be pinpointed precisely because many threads are replied to with the message that 'the problem has been added to the bugtracker' or 'we will try to fix the problem in version 1.5'. Certain issues the author has observed personally have not been fixed properly even after a few months.

Because no indication that ATutor has serious security problems was found, a minimal score is not given. However, the score will not be very high due to the obscurity of the bugtracker, the lack of activity and high response time.

I.7.1 Score

Moodle: 9

ATutor: 4

I.8 Functionality

The feature lists of the two applications do not give a conclusive answer in this case. ATutor and Moodle each have their own philosophy and thus a different approach to describing their application. The choices of what features to describe and which types not to describe also differs. Apart from that, the Moodle feature list is self-admittedly not complete: '*Moodle is an active and evolving product. This page lists just some of the many features it contains*', and some of the included modules are not described at all. Therefore it is not enough to put the feature lists side by side.

Both applications were installed on a test-environment to explore the functionality.

The list of required functionality defined in Chapter 4 is used here.

- Course based
  - Basic course information
  - File sharing
  - Communication
  - Groups
  - Tests and quizzes
- Access control
- Administration
- Language

For both applications, each of these items were evaluated to see to what extent and how well it has been implemented. In addition, any additional functionality that may prove useful in the University situation was taken into account.

I.8.1 Moodle

Moodle is course based, and also supports global items that are available for all users on the main page.

General information is set in the course settings and can be added to the main page using a block[6]. Announcements can be posted on the main page using the News forum that is created automatically for each course. Lecturers can be found in the participants list where the user can view the profiles of the users in the course.

The courses can be organised in different ways. There are three course formats included: Weekly, by topic and discussion focused. The first two allow different sections for each week/topic where each section can contain any number of resources and activities that Moodle has.

File sharing: The lecturer can upload files in the course area and link to them from different sections of the course page. Students can attach files to several resources, such as assignments and wiki pages.

Communication: Forums can be created to allow discussions. Just like all the other activity modules, the forum can be set for different levels of group access, meaning they can be global to all users, or separate forums for each group, with or without the other groups being able to see each other's forums. A chat module can also be added to allow real time communication between students with the same access possibilities as mentioned above. Past chat sessions can be viewed. Access to these sessions can be set by the teacher.

Groups can be created by the teacher, who can also add the students to the groups. The teacher can also allow the students to assign themselves to groups. Because group access levels can be set for each activity, group work can be done in many different ways.

Tests and quizzes: The quiz activity allows a teacher to define a quiz with several types of questions, such as multiple choice, short answers, and numerical. Different types of grades are also available, and the grading can be either automatic or done by hand.

Access control can be accomplished through many different ways. Moodle supports authentication using the following methods and/or protocols:

- LDAP – Lightweight Directory Access Protocol, used by many universities and in other large networks for authentication
- IMAP / POP3 / NNTP – Mail servers. If the users all have mail accounts on a central server, authentication can be done using that login information
- External Database – defining a database table with the users' information
- Manually – users register themselves by filling in a registration form

[6]Blocks are separate pieces that can make up a page, usually can be positioned in different places, for example in a three column layout, like Moodle, can be positioned on the left, middle or right of the page and moved up and down, switching positions with other blocks. A block can contain links, text or other information, like a summary of news items

Various other authentication methods are being added or have been added.

Access control to different parts within Moodle is done by assigning roles and by defining levels of access. For example, a teacher can maintain courses, a course creator can also create them. The required access level can be set for each course, in terms of guest access (allowed or not allowed), and optionally using an enrolment key, which can be a word or phrase that the student needs to enter in order to get into the course.

An administrator can define all the settings of the Moodle site as well as perform all the other tasks such as creating and maintaining courses. On a global level the administrator can change many settings, such as the authentication method, edit users, assign roles, set the global theme (look and layout of the site), set the default language and what other languages are available and so on.

Moodle has 61 languages packages available (Moodle, 2005f). The administrator can define which of these are available in the installation. The user can choose which language he or she wants to use on the global site, and for each course a default language can be set and this languages can optionally be 'forced', meaning the user cannot override the language setting, so the course interface will always be in one language only.

Additionally, Moodle has a number of extra features. Moodle offers modules that can be used per section or globally in each course. These consist, apart from the features mentioned above, of the following:

- Assignments – where students can hand in documents and such that can be graded.
- Glossary – where terms can be defined either by the teacher or also by the students. The settings can allow a term to be defined more than once or only once. The glossary can be one of several different formats, such as encyclopedia, dictionary and FAQ.
- Lesson – a certain way of presenting content to the students, with pages and intermittent questions where the answers can define whether a student can move on the next page or not. The lessons are very customisable with many settings and options
- Wiki – a wiki is a type of content system where users can contribute in a very straightforward way by editing pages. A well known example is wikipedia, an online encyclopedia maintained by anyone who has something to add. This type of tool is discussed much lately in terms of collaborative learning. The wikis, once again like any other module, can be used on a per-group basis or globally.
- Workshop – a peer assessment activity allowing participants to assess each other's projects

These are the modules that are supplied with Moodle by default. However, the structure of Moodle allows modules to be added very easily. Instructions are available on the Moodle website how to add a module and what its structure should be. There are additional modules available at the Moodle website (Moodle, 2005g), including Dialogue, Flash (for flash movies), Book and several others, that are either contributed by other users or are still in development status.

Apart from the modules, there are numerous other additional features, such as user logging and tracking, a calendar listing site events, course events and user events, a backup facility to create backups of courses and the entire site, mathematics tools and a HTML editor for most text fields.

I.8.2 ATutor

ATutor is course based. General information on the course can be put into announcements on the main page. It does not seem to have any separate sections in which the user can get information on the teachers.

The teacher can upload files through a file manager and link to them in other course content.

By using the ACollab add-on for group functionality, groups can share files through their group area.

Communication: The teacher is able to send email to students through ATutor. With ACollab groups can chat and use a forum for communication.

Groups can be created through the ACollab interface by the teacher. The teacher can add users to the groups. The group interface allows users to use forums and chat, as well as share a calendar. There is also a drafting room for sharing documents.

Tests and quizzes: The teacher can create tests and surveys, with several types of questions such as multiple choice, open ended, and true false. These tests can be self marking. The test can be released for a certain period.

Access control – for the moment it appears that ATutor only has the ability to use manual registration for authentication. According to some forum posts they are working on integrating LDAP into ATutor.

The ATutor administrator can manage existing users, courses, assign instructors to courses, create course categories and define the site's language and theme. The administrator can also start a new translation in ATutor itself and later donate this translation back to the ATutor project for others to use.

There are 15 languages available for ATutor (ATutor, 2005b). The administrator can install these language packs on the server. This can be done through a local file or directly from the ATutor website. The preferred language can be chosen by the user and changed at any time. The primary language for a course can be set by the instructor.

ATutor has some additional functionality, namely a course glossary, content export, backup manager and an enrolment manager to import course lists and manage user privileges.

I.8.3 Overall

When comparing the features of Moodle and ATutor it appears that Moodle's features are very rich where most of ATutor's are more basic. Also, additional functionality offers much more in Moodle than in ATutor. It seems that ATutor's focus on accessibility takes away some of the attention from enriching and adding of features. Some technical issues came up during installation of ATutor and ACollab, while Moodle

installs and upgrades smoothly time and again. Moodle appears much more comfortable and intuitive to navigate and work with than ATutor.

I.8.4 Score

Moodle: 9

ATutor: 6

I.9 Integration

Integration of software is accomplished in several ways.

I.9.1 Modularity

Modularity enables easier integration of existing code. When a modular structure is used, other modules can be added easily. Moodle has such a structure for activities, as well as themes, which define the look of the site, languages, and in a lesser extent for course formats.

ATutor does not appear to have such a modular structure to allow addition of features in the same manner.

I.9.2 Standards

Official and de facto standards exist for many different types of applications. In the E-Learning sphere the most important standard is actually a collection of standards, named SCORM (Sharable Couseware Object Reference Model) consisting of several IMS standards for content structure and packaging and some other standards and guidelines for content, packaging, sequencing and navigation.

Both ATutor and Moodle have SCORM compatibility in that they can 'play' SCORM packages.

I.9.3 Compatibility with other applications

In this case, compatibility with other software will mostly be relevant in terms of document processing and such. Moodle has some filters available that allow, for example, to show mathematical equations and TeX[7] documents. ATutor does not appear to have such compatibilities.

In terms of authentication, as seen with functionality, Moodle has a large number of options, compatible with systems such as mail servers and LDAP, while ATutor currently does not support other authentication methods.

Moodle can be optionally integrated with several other web applications such as Content Management Systems, for example Mambo, Xoops and Postnuke, web hosting control panels that allow for simple installation of applications like Moodle, such as

[7] typesetting system popular among academia (Wikipedia, 2005c)

CPanel and Plesk, and Moodle is also available as a Debian package, allowing for easy installation on Debian Linux systems.

Software Requirements

Both ATutor and Moodle run on PHP. ATutor is a little more restrictive in its requirements:

Moodle (Moodle, 2005e):

1. Web server software. Most people use Apache, but Moodle should work fine under any web server that supports PHP, such as IIS on Windows platforms.
2. PHP scripting language (version 4.1.0 or later). PHP 5 is supported as of Moodle 1.4.
3. a working database server: MySQL or PostgreSQL are completely supported and recommended for use with Moodle.

Moodle also names the flexibility in the developer documentation: *'Moodle should run on the widest variety of platforms'* (Moodle, 2005c).

Atutor (ATutor, 2005a)

1. HTTP Web Server (Apache 1.3.x is recommended. Do not use Apache 2.x).
2. PHP 4.2.0 or higher with Zlib and MySQL support enabled (Version 4.3.0 or higher is recommended).
3. MySQL 3.23.x or higher, or 4.0.12 or higher (MySQL 4.1.x and 5.x are not officially supported).

Score

Moodle: 10

ATutor: 7

I.9.4 Total Score

Moodle: 9

ATutor: 5

I.10 Goal and Origin

Moodle's origin is clearly explained in the 'Background' page of the documentation (Moodle, 2005a). As discussed in Chapter 4, it was started as part of a PhD research project. The philosophy is included in the documentation as well, explaining the social constructionist pedagogy on which the design of Moodle is based. The development documentation also explains some of the goals in terms of programming and

compatibility, focussing on a wide compatibility, easy installation and upgrade, and modularity. The future plans are depicted in the roadmap (Moodle, 2005i), showing the proposed feature additions for the next three versions.

Apart from ATutor's clear statement to focus on accessibility, information on ATutor's background was not found, in terms of the motivation of the developers and for whom the project was created and is being developed.

The ATutor roadmap (ATutor, 2005e) gives a general impression of future plans, such as the addition of add-ons and content packaging options.

I.10.1 Score

Moodle: 10

ATutor: 5

I.11 Overall observation

When trying to establish the information above about both projects, a significant difference was observed, mostly in information availability. Where for ATutor the author had to search for much of the information and was not able to find out certain things, like the project's origin and the goal of the developers, while the Moodle website offered information in abundance, including links to articles of the research project that got Moodle started. Also, the Moodle project has a very welcoming community feel to it, where the visitor is encouraged to participate at every turn, where ATutor seems more distant in that respect. Though the developers of ATutor may certainly not be unwilling to allow users to participate, the ATutor project's website does not do much to encourage that.

Appendix J

Remarks on validity of case study results

The top five from the selection result was:

- Moodle
- ATutor
- ILIAS
- Claroline
- Dokeos

The top two of these were evaluated in depth. Moodle came out very strong.

Each of these systems is discussed briefly below in terms of their real-life performance, in reverse order.

Dokeos and Claroline Dokeos originated from Claroline, when a number of developers left the Claroline project in the summer of 2004 and started the Dokeos project using the Claroline code base. These systems do not differ enough at this time to investigate them separately.

The Vrije Universiteit Brussel (VUB) has chosen to implement Dokeos institution-wide under the name PointCarré, replacing their Blackboard installation. They evaluated a short list of Claroline/Dokeos, Moodle and Blackboard. In part because of location (the Claroline/Dokeos project team resides in Belgium, like the VUB), the choice was made in favour of Claroline/Dokeos (VUB, 2004). They are now working closely with the developers.

ILIAS ILIAS is created by the University of Cologne, Germany. It is used there institution-wide (ILIAS, 2005).

A comparison report by the Commonwealth of Learning (COL)[1] compared Open Source CMSs, with a candidate list of 35 and a short list of 5: Moodle, LON-CAPA, ILIAS, dotLRN and ATutor. ILIAS came in second in this report (COL, 2003).

[1] `http://www.col.org`

ATutor ATutor was the top system in the COL report (COL, 2003).

Another article from 'Progress through Training,' a training centre in the United Kingdom, compared Moodle, Claroline and ATutor. It was published in August 2003, and recommended ATutor as first choice (Clements, 2003).

Moodle Moodle was the CMS of choice for Dublin City University last year when they evaluated CMS options. It is now used there institution-wide. This decision was made in an evaluation process, with a short list of Bodington, Claroline and Moodle (McMullin and Munro, 2004).

Moodle was shortlisted in the COL (2003) report and the Clements (2003) report. It is suspected it was not recommended at that time because its features were not as elaborate as they are now.

Moodle has gotten coverage by many technology-oriented news sources, as well as Linux Journal and a large number or research papers, all of which can be found through the 'Moodle Buzz' page (Moodle, 2005b).

Moodle has also been chosen by a joined effort of two E-learning projects in New Zealand. 'NZ Open Source VLE project' [2] and 'The Open Source Courseware Initiative New Zealand' [3] have chosen to use Moodle after comparing several systems (Ose.Nz.Org, 2004).

[2] `http://ose.nz.org`
[3] `http://www.elearning.ac.nz/`