From Open Source to Open Content

Creating an Information Model for Open Source Software

Sirpa Alanko
University of Tampere
School of Modern Languages and Translation Studies
English Philology
Pro Gradu Thesis
May 2008

Tampereen yliopisto
Englantilainen filologia
Kieli- ja käännöstieteiden laitos

ALANKO, SIRPA: From Open Source to Open Content: Creating an Information Model for Open Source

Pro gradu -tutkielma, 115 sivua
Toukokuu 2008

Tämän Pro Gradu -tutkielman päätavoitteena on tarkastella sisällönhallintajärjestelmien merkitystä projekteille, joissa tuotetaan avoimeen lähdekoodiin perustuvia ohjelmistotuotteita. Avoimen lähdekoodin projekteissa kehitystoiminta perustuu pitkälti tuotteen, projektin ja prosessien läpinäkyvyyden ja avoimuuden edellytyksiin. Nämä mahdollistavat parhaimmillaan äärimmäisen nopean ja tehokkaan kehitystyön, jota suorittavat usein samanaikaisesti useat vapaaehtoiset osallistujat ympäri maailmaa. Vaikka menestyksekkäitten projektien taustatekijöitä onkin jo pyritty kartoittamaan useissa tutkimuksissa, nämä tutkimukset keskittyvät useimmiten ohjelmistokehitystä, eivätkä juurikaan tarkastele dokumentaation tai sisällönhallinnan merkitystä tai avoimuutta.

Tutkimuksen keskeisenä lähtökohtana on oletus, jonka mukaan em. projekteihin liittyvä sisällönhallinta ei ole yhtä avointa kuin vastaava ohjelmistotuotanto. Tämän puolestaan oletetaan olevan ensisijainen syy siihen, miksi avoimen lähdekoodin dokumentaation ei yleisesti koeta täyttävän sille asetettuja vaatimuksia em. ohjelmistotuotteiden tavoin. Tutkielmassa verrataan ensin dokumentaation ja lähdekoodin avoimuutta rinnastamalla ne datan, informaation, tiedon, sisällön ja toiminnallisuuden käsitteihin. Tämän lisäksi lähdekoodin avoimuuden mahdollistavia tekijöitä sekä edellytyksiä kartoitetaan kirjallisuuskatsauksen avulla.

Tutkielman toinen päätavoite on kehittää yleisen tason viitekehys avoimen lähdekoodin projektien sisällönhallinnalle, joka loisi edellytykset avoimuuden vaatimuksen täyttämiselle. Tutkimuksen teoreettisen viitekehyksen pohjana on Hackosin 2002 julkaisema sisällönhallinnan informaatiomalli. Tutkimuksessa määritellään käyttäjien tiedontarpeisiin perustuvan informaatiomallin kolmen pääulottuvuuden (käytön ulottuvuudet, informaatiotyypit, sisältökomponentit) keskeiset käsitteet, joita käytetään julkaistavaa sisältöä rakennettaessa, organisoidessa ja nimettäessä. Tutkimuksessa keskitytään informaatiomallin luomisen kahteen ensimmäiseen vaiheeseen: käyttäjien tarveanalyysiin sekä itse informaatiomallin ja sisällönhallintajärjestelmän toiminnallisten vaatimusten dokumentoimiseen.

Dokumentaation ja lähdekoodin vertailu käsiteanalyysin avulla osoitti, että nämä kaksi informaation muotoa eivät ole käyttäjilleen yhtä yksiselitteisiä tai avoimia eivätkä kykene välittämään tietoa ja osaamista samalla tavoin. Sisällön ja toiminnallisuuden (joista ohjelmistotuote koostuu) käsitteiden ominaisuuksissa sen sijaan havaittiin samankaltaista avoimuutta. Tutkimuksen johtopäätöksissä esitetäänkin, että perinteisten dokumenttien tuottamisen sijaan avoimen lähdekoodin projekteissa tulisi pyrkiä metadatan avulla tunnistettavan sisällön tuotantoon ja tietovirtojen hallintaan.

Informaatiomallin keskeisiksi ulottuvuuksiksi todettiin sisällön luokittelu tuote- ja projektikohtaisiin sisältöihin, jotka on kohdistettu projektin erilaisille osallistujaryhmille. Sisällönhallintajärjestelmän ja ohjelmistokehityksen välineiden ja prosessien vaatimuksissa havaittiin useita samankaltaisuuksia. Lisäksi havaittiin useita tekijöitä jotka viittaavat sisällönhallintajärjestelmien olevan keskeinen tekijä avoimen lähdekoodin projektien menestykselle.

Avainsanat: avoin lähdekoodi, avoin sisältö, dokumentaatio, sisällönhallinta, informaatiomalli.

Acknowledgements

Therefore, since brevity is the soul of wit,
And tediousness the limbs and outward flourishes,
I will be brief.

William Shakespeare: *Hamlet* (Act 2, scene 2, 86–92)

I would never have completed this work without the support of my mother. I also wish to thank Tytti Suojanen for her guidance and excellent advice during the process of writing this thesis. Last, but certainly not least, I want to mention my dog, Hupi, for being there to remind me that the impossible *can* happen.

I wish to dedicate this study to the memory of my father.

Tampere, 14 May 2008

Sirpa Alanko

Contents

List of Figures

List of Tables

1 Introduction

The wide success of Free/Open Source Software (F/OSS) has recently attracted much attention. For example, on 16 January 2008 the news headlines all over the world revealed that Sun Microsystems Inc. has agreed to buy open source database software developer MySQL AB for $1 billion. It is apparent that open source software has become a mainstream part of the market and that both end-users and the corporate world is seeing open source as a viable option. So let us have a closer look at what the fuss with F/OSS is all about.

First, however, it should be noted that many definitions about F/OSS exist. In fact, one can even divide the movement into two different models of software development: free software vs. open source software. While it is not in the scope of this study to discuss and define the evolution of the F/OSS phenomenon exhaustively, some more background related to the open source movement will be given later in Chapter 4. In this study, the term OSS will be used from this point onwards to refer to the phenomenon being studied.

Apart from the "strictly business" side of things, researchers and commercial companies alike are trying to learn lessons from the success of OSS and even apply some of the success factors to the development of proprietary and closed systems. For example, researchers involved in a multi-disciplinary research project called OSSI (Managing Open Source Software as an Integrated Part of Business) have pointed out how:

> Companies [...] want to understand the OSS phenomenon to be able to make the decision whether to be involved in OSS or not. However, there is also another, perhaps more recent reason behind the eagerness to understand the logic and practices of OSS – the desire to learn from OSS development in order to apply the best OSS practices in other contexts as well. For a software company, for example, [the] important question is that what are the best OSS practices and how could we apply them in our software development and business? (Helander and Antikainen 2006, 1)

Ye and Kishida (2003) define OSS as "those systems that give users free access to and the right to modify their source code." According to Robertson (2004b), having access to all the source code allows local developers to make any required changes to the system to meet specific business

requirements. Furthermore, the most popular open source products are supported by a community of hundreds, if not thousands, of developers while little community typically exists around commercial solutions where communication and information sharing only occurs between customers and the company's support staff. Thus, when a bug is identified in a commercial solution, all you can do is to report it to the vendor, and wait for them to fix it. With an open-source product, you can try reporting the issue to the community, which often helps identify a patch or workaround in only a number of days. Alternatively, you can solve the problem yourself: with full access to the source code, there is no issue that cannot be resolved if you possess the required knowledge.

Interestingly enough, open source documentation is often a different matter altogether. For example, a simple Google search using the words "open source documentation" results in numerous accounts about just how bad the documentation can be. Among others, Mork (2006, iii) has pointed out that "[o]pen source has a reputation of creating high quality software, but documentation of process and product is weak". A quick assessment of the aforementioned Google search results implies that usually the case is that hardly any documentation exists, or the information is inaccurate, outdated, poorly organised or irrelevant to the user. One might even argue that in many cases documentation seems to be the Achilles' heel of OSS. There are exceptions, of course, and, moreover, it also seems that the situation is gradually changing as bigger players are venturing into the world of open source.

But why is it that the open source documentation often does not seem to meet the users' expectations the way OSS code does? Why is an open source community unable to create documentation in as efficient and flexible manner as it produces new features and bug fixes to the code? This question raises several others, to which it is by no means any easier to find a simple, definitive answer:

1. What are the central success factors behind a popular Open Source Project (OSP) and what role does documentation play in this success? When trying to answer this question, it is

important to begin by noting that a great deal of controversy exists within the OSS communities about the importance of formal documentation.

2. What information should be included in OSS documentation for it to meet the needs of its audience and turn it from bad to good? What are the target audiences of OSS documentation and what are their needs?
3. What exactly constitutes OSS documentation? In other words, what is really meant — and what *should* be meant — when referring to *open documentation*? Should some other term be invented and used instead? When it comes to information sharing, it seems OSS is breaking the mold just as it has done with software products and code. For example, the open source philolophy has inspired the creation of new licences, concepts, and projects such as community authoring, Wikipedia[1], LIFE OpenContent[2], open knowledge[3], just to name a few.
4. When we try to share information using the same principles that we use to share OSS code, are there some crucial aspects of the process that we ignore or neglect that might in turn account for the argued poor quality of OSS documentation and the lack of contributions to it? Or is information/documentation an altogether different kind of beast that cannot be developed and shared in the same way as OSS code?

One of the most popular explanations for the poor quality of OSS documentation stems from the now famous remark made by Raymond (2001) "Every good work of software starts by scratching a developer's personal itch." An open source project typically starts with a developer trying to solve a personal problem. Thus, s/he focuses on what s/he finds interesting, that is, coding. As the developer knows perfectly well what s/he is doing, there is no need to scratch someone else's itch by writing documentation. The following samples from the SlashDot discussion forum portray well the general attitude of open source developers when it comes to documentation.

1. http://www.wikipedia.org
2. http://www.life-open-content.org
3. http://opendefinition.org/

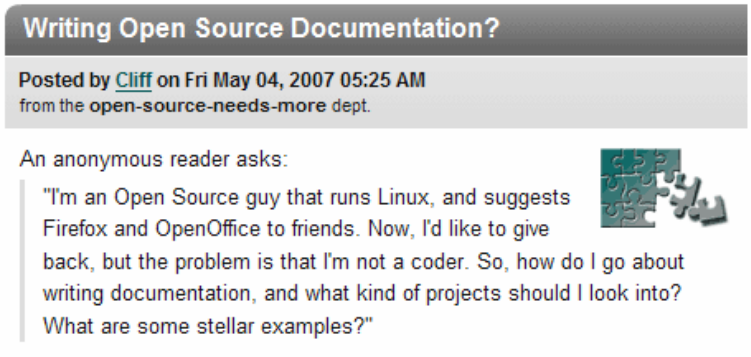

Figure 1. Question by an anonymous reader at http://ask.slashdot.org/

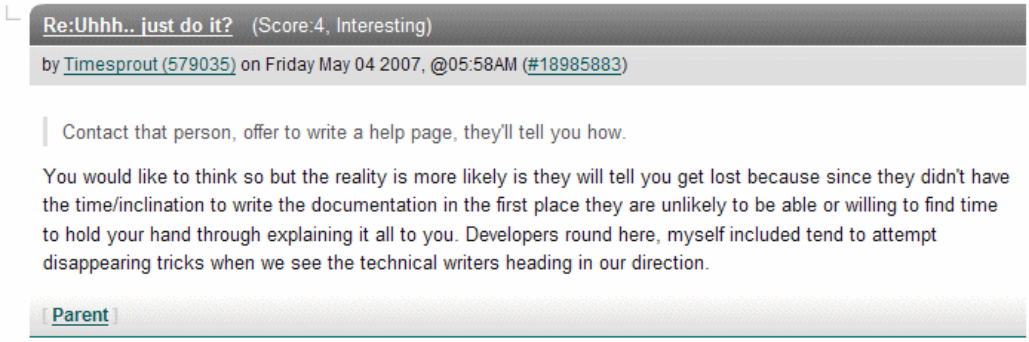

Figure 2. Answers to the question shown in Figure 1 at http://ask.slashdot.org/.

Figures 1 and 2 also demonstrate how difficult it can be for outsiders to contribute documentation. While open source projects warmly welcome user contributions to create and/or improve the documentation just as is done with the codebase, documentation often proves to be a daunting task. What makes the situation even more interesting is that open source projects often suggest that newbies (with no or very little knowledge about the project or the product) as their first contribution to the project start writing documentation to educate others (Tyler 2006). This reveals something not only about how high (or low) documentation is rated on the list of OSS success factors but also how documentation is sometimes regarded a somewhat menial task that requires less talent, intellect, and expertise.

Fortunately, not everyone feels this way about documentation, as is shown by the following comment posted by an anonymous OSS user:

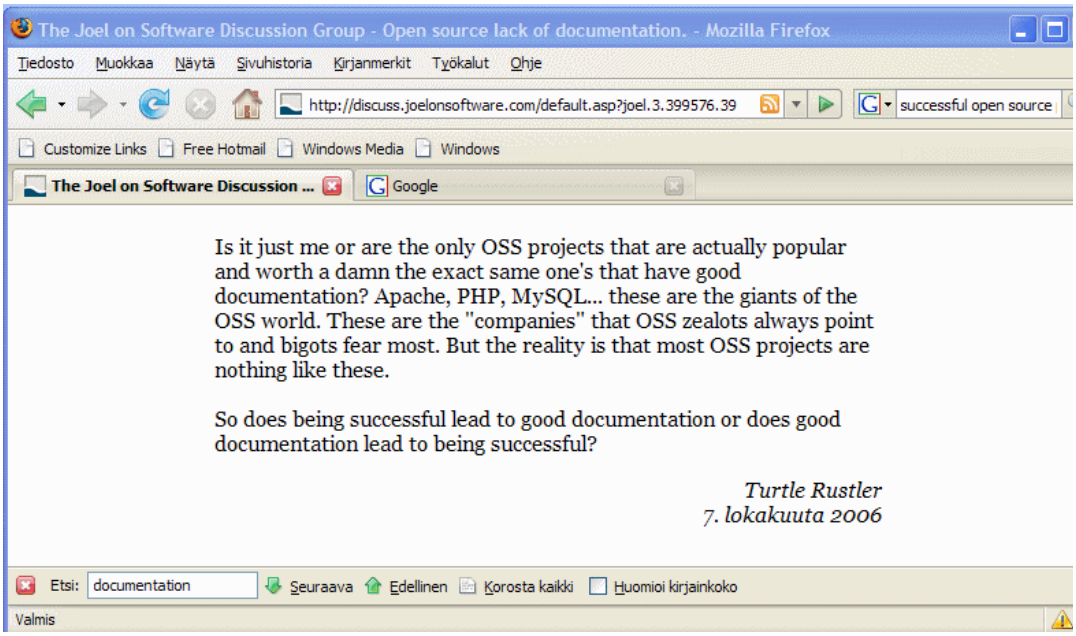

Figure 3. Comment by an anonymous user at http://discuss.joelonsoftware.com/

To consider the importance of documentation for open source, I will present yet another real-life example taken from a blog entry. Fred Ingham, after spending a few weeks evaluating two open source applications, describes his frustrating experience as follows:

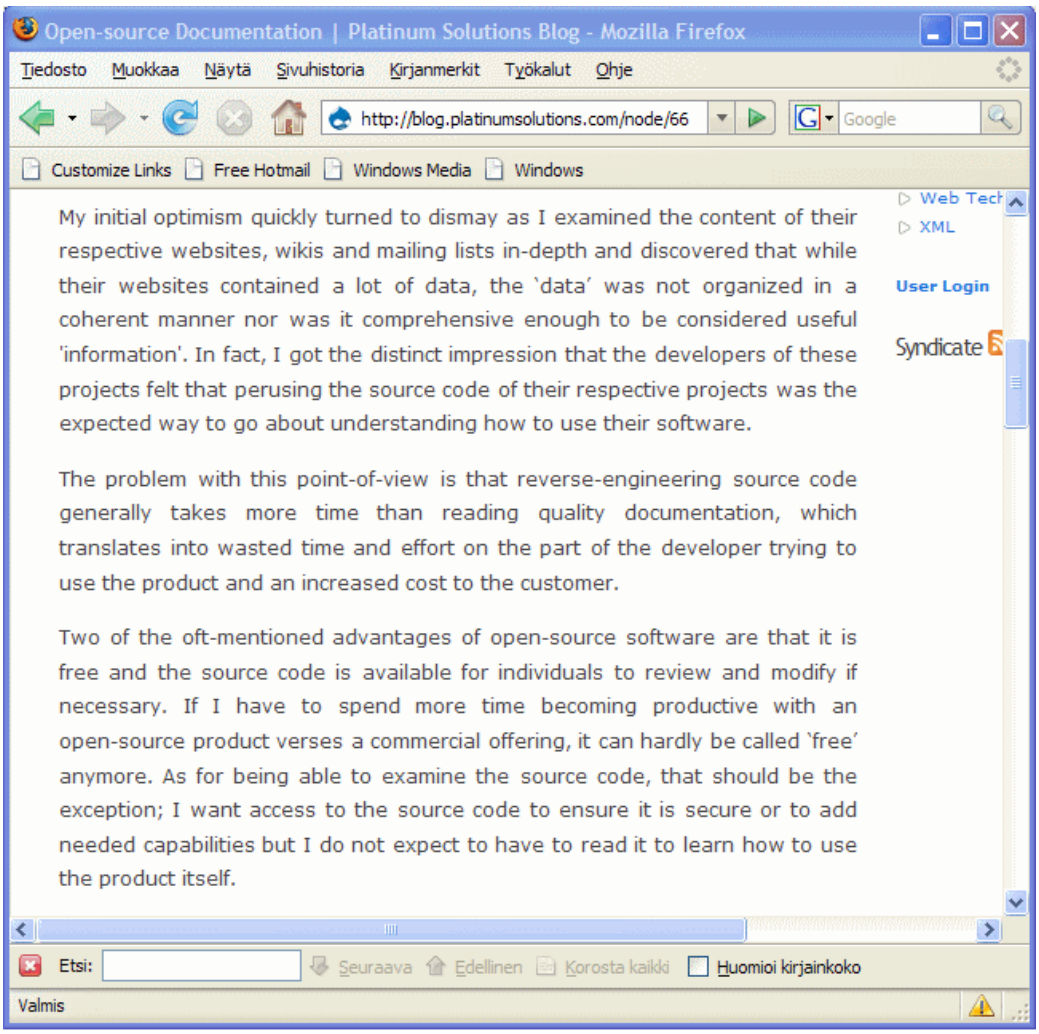

Figure 4. Fred Ingham's blog at http://blog.platinumsolutions.com/node/66

The example in Figure 4. suggests that end-user documentation at least is a central success factor for OSPs, and that poor documentation can even hinder the adoption of open source software. Furthermore, the example also demonstrates a need for identifying ways to improve open source documentation.

1.1 Purpose of the study

The main purpose of this study is to assess whether the documentation of Open Source Projects (OSPs) remain more or less "closed source" to the OSS community. In other words, my current hypothesis is that open source documentation does not fulfill the requirement of *openness* the way open source code does. I aim to show that to achieve such openness, an OSP requires the

creation and sharing of an *information model*, that is, a framework that forms the basis for the OSP's *Content Management* (CM) and *Information Architecture* (IA). Morville and Rosenfeld (2007, 11) provide a useful definition of information architecture and its relation to content management:

> Content management and information architecture are really two sides of the same coin. IA portrays a "snapshot" or spatial view of an information system, while CM describes a temporal view by showing how information should flow into, around, and out of that same system over time.

I believe that the lack of a community-based information model is one of the main hindrances and obstacles that prohibit OSS documentation from being developed as efficiently as open source software. I also presume that this is one of the main reasons why open source documentation may fail to meet the needs and expectations of its audience. In fact, instead of approaching this dilemma from a documentation-specific point of view, researchers and OSS experts should aim at producing *open content* and a comprehensive information model for open source.

I will assume the role of an *information architect* and attempt to demonstrate that the basic principles of open source development and development of open content are — if not identical — at least similar to a great extent. If this is true, it would be a great controversy if the know-how and intelligence behind an open source information model were not shared and developed in a way similar to that of the open source software that is being documented.

According to Hackos (2002, 343-344), an information architect is chiefly responsible for the information model. An information architect must be able to "analy[s]e business, authoring, and delivery requirements and mold these into a vision of the user's experience of the future and an outline of the workflow scenarios that will have to be supported by process and technology". In other words, the architect must "balance the needs of users with the goals of the business" (Morville and Rosenfeld 2007, 5).

I will also provide some ideas about how opening up the information models of OSPs to the open source community might perhaps revolutionise the field of technical communication as the open source movement has done to Information Technology (IT). I will present some examples

about areas where we should try to breach the gap between software architects and information architects. I will discuss some of the advantages that might result from making it clear we are in effect striving for the same goal. Moreover, I will discuss some characteristics of the open source ideology that could be expanded to the field of technical communication to overcome or at least mitigate some of the major challenges faced by information architects and authors of technical documentation today.

The second main goal of this study is to provide a general-level OSS information model that might provide a starting point in the development of open content for both existing and new OSPs. I acknowledge the fact that this information model will be far from complete: thus another important aim of this work is to identify areas that require further study. Furthermore, as a part of designing an OSS information model, I will also briefly discuss the phenomenon known as *community authoring*.

1.2 Background to the study

The idea for this study evolved during a number of years when I have been working as a technical writer or information designer. The most difficult questions that I have been facing time and time again in my work have always been related to the design of the information architecture, that is, the content plan for documentation that is delivered online. I have been struggling to find an answer to questions such as how I should structure and organise the publication that I am creating, what type of information is relevant and what is irrelevant to different types of users, and what categories and headings I should use to make my information architecture easily comprehensible and accessible to different users. To help readers recognise the documents or sections that deserve their attention, the documentation should be structured so that the main ideas catch the attention of the readers, this being all the more important if the same information is used by different groups of users with different needs.

But how can I actually achieve this? Even when working in a project where there was a user and task matrix available, applying the information in the matrix to build the information architecture was far from obvious. Furthermore, my experiences are further validated by Salvo (2004, 39-40) who argues that

> [t]echnical communication research describes a variety of analytic methods for collecting, assessing, and representing data and turning these data into usable information. But researchers have not offered strategies for moving from analysis to action—for putting the hardwon information to use and enacting strategies for action that meaningfully engage the world.

It was not until I started working as a technical writer for a company whose proprietary software products are based on an open source technology that I started finding answers to some of my questions. The company's open source portal included mailing lists for the hosted open source projects, which I followed regularly to gather information both about the product itself and also the information needs and usability problems of the users and/or developers themselves. This was the closest I had ever got to real end-user experiences during the seven years I had spent working in the field of technical communication. Based on conversations with my colleagues, many technical writers are still forced to make educated guesses about the needs of their target audience(s). The following statement by Berglund and Priestley (2001, 140) is a very apt description of my experience:

> [...] [open source] users definitely can provide questions even when they can't provide answers. In this sense, open-source documentation provide[s] much needed relevance and priority assessments to the documentation process.

On the other hand, one can find only a minimal amount of research looking at the open source phenomenon specifically from a documentation or content management point of view. Furthermore, while it is true that each documentation project should be evaluated case by case and that the quality requirements, methods, and tools used must be adjusted to suit the current situation, requirements, schedule etc., most documentation projects do not have enough allocated resources to conduct a full-blown user analysis to identify and define the audience needs. This holds most certainly true for new OSPs that are nowadays sprouting like rabbits[1]. Consequently,

there is definitely a need for both general information models and more detailed case studies that can be used as a starting point by authors and information architects working in projects of similar function, scope, or target audience. Open source documentation is also an important subject for study because of its contemporary nature:

> The way we educate ourselves to use and program computers is shifting along many of the same historic lines as journalism, scientific publication, and other information-rich fields. Researchers have pounced on those other trends, but computer education remains short on commentary. [...] This [community authoring] movement cuts into my living as an editor of conventional documentation, for several reasons I desperately need to understand. (Oram 2006)

As a technical communications professional, I am interested in this phenomenon for very much the same reasons.

1.3 Theoretical framework

As the main theoretical framework for my study I will use Hackos' book *Content Management for Dynamic Web Delivery*. The book, published in 2002, provides reasonably fresh insight into content management implementations and the information models behind them. Moreover, Hackos stresses the importance of a *community-based* information model. Hackos' web-delivery-focused approach is also in line with the argumentation of Berglund and Priestley (2001, 135), according to which "an absolute requirement for open source documentation is the electronic format".

Hackos (2002, 36-49) divides the content management process into the following five phases depicted in the figure below:

1. needs assessment
2. writing the information model and outlining the functional requirements of the CMS
3. creating the content assembly and delivery plans based on the information model
4. conducting and evaluating a pilot project
5. rolling out to the larger enterprise.

1. For example, on 13th May 2008, there were 177,014 registered projects at http://sourceforge.net/

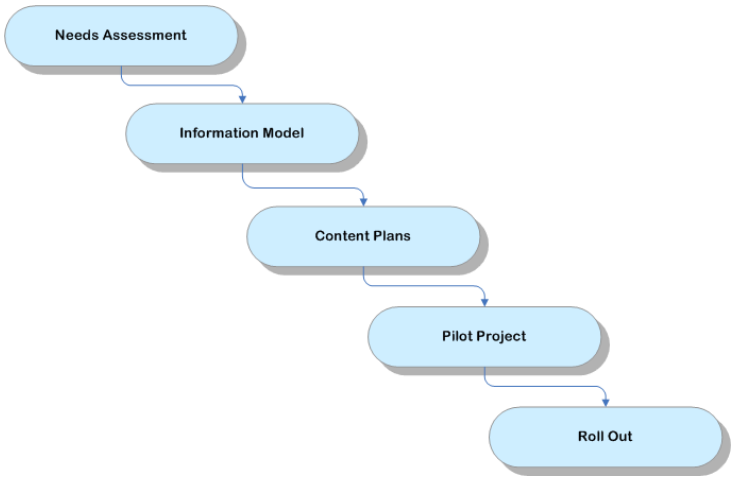

Figure 5. The recommended workflow for the content management project (Hackos 2002, 36; 338)

In this study I will only cover the first two phases of the content management process: as the purpose of the study is to build a general-level information model for OSS, it is impossible to create detailed content plans as the information covered in such plans would need to be project-specific. The deliverables of **Phase 1: Needs Assessment** include a report and a recommendation, which:

- define "the business problem at hand" and how the organisation will benefit from the new system
- specify the business case to "show what it costs to continue handling content as it is done today, what the short-comings are of the current approach and what efficiencies and cost savings might be reali[s]ed with a new and better solution". (Hackos 2002, 38)

Phase 2: Information Model will produce the following deliverables:

1. An information model (or several interrelated information models).
2. A functional requirements document, which is based on all the information gathered thus far (i.e. the needs assessment and information model(s)).
3. A guideline for authors for implementing the information model. (Hackos 2002, 39-43) This deliverable is not included because the purpose of this study is to include a general-level information model for open source.

Where relevant, I will modify Hackos' content management model based on OSS research so that it can be applied to the OSS world. I will broaden and compliment Hackos' theories about content management and information architecture with those presented by Boiko (2005) and Morville and Rosenfeld (2007), among others.

1.4 Organisation of the study and material and methods

This study consists of the following parts:

In Chapter 2 I will perform a conceptual analysis where I aim to disintegrate the concepts of *open source code* and *documentation* into the very basic units of human communication and understanding, that is, *data*, *information*, *content*, *knowledge*, and *wisdom*. The purpose of the conceptual analysis is to better allow the comparison of *code* and *documentation* in order to determine whether they can be shared, transferred, and reused as openly. The analysis will also appraise the function of content management and information models for the achieval of such openness.

Chapter 3 presents Hackos' (2002) three-dimensional information model. Chapter 3 also aims to clarify the previous, rather abstract content analysis of *information* and *content* by giving simple examples of how we unconsciously handle information types and content units — the basic building blocks of content management — in our everyday lives.

Chapter 4 represents the first phase of the content management process, needs assessment, as defined by Hackos (2002). Thus, Chapter 4 lays the foundation for creating an open source information model. I will perform the needs assessment based on existing, relevant OSS research, which provides information regarding, for example, OSS management frameworks and OSS communities, and which is therefore also valuable for the field of technical communication and content management. If and when no answers can be provided by the existing studies, I will document their absence under suggestions for future research. In addition, I will discuss the

definitions and characteristics of OSS and describe how OSS research defines the concept of openness.

In Chapter 5 I will build on the discussion included in the previous three chapters and use Hackos' approach to create several interrelated information models as a part of defining a content management framework for open source.

In Chapter 6 I will first describe the anatomy of a CMS and then relate the discussion to the functional requirements of an open source CMS.

Chapter 7 presents the conclusions of the study.

2 Defining data, information, content, knowledge, and wisdom

In this chapter I will define what is meant in this study by the interrelated, abstract, and often fuzzy concepts *data*, *information*, *content*, *knowledge*, and *wisdom* to be able to examine and evaluate:

- what is the essence of open source code and documentation, i.e. how they relate and/or correspond to the concepts of *data*, *information* and *content*
- what is the essence of *knowledge* and *wisdom* and if and how they can be captured, managed, or transferred using *data*, *information*, and *content*
- what role do open source code and documentation play in the transfer of information and/or knowledge taking place in an OSP
- what is the significance of a Content Management System (CMS) that is based on an information model for information and/or knowledge transfer and also the openness of an OSP.

As has been pointed out by Hey (2004, 2), the concepts of *information*, *knowledge* and *wisdom*, not to mention the transitions between them, "still resist clear definition". The fuzziness or even obscurity of these concepts is by no means diminished when estimating if and how *knowledge* or *information* can be captured, managed, or transferred through *information*, *knowledge*, or *content management*. It is imperative for the purposes of this study to establish what is meant here with the concepts listed above. They are at times used interchangeably and/or synonomously in the literature which forms the theoretical background of this study. Thus, I will establish what the concepts are to which I am referring when using these words, and, in some cases, explain the justification behind modifications made to the quotations included in this study.

I will perform a conceptual analysis of these terms and their definitions and use the results as a part of my theoretical framework to compare the openness of code and documentation sharing in OSPs: I will break down the concepts of *open source code* and *documentation* to the level of *data, content* and *information* to allow the comparison.

2.1 From data to information and knowledge

Hey (2004, 12) defines *information* as data with meaning. In other words, *data* is raw material that must be processed, shaped, and structured to become information. According to Boiko (2005, 7-8), for information to exist, a human being first has to:

1. form a mental image of a concept that s/he wants to communicate to someone else
2. use intellect and creativity to choose words, sounds, or images that suit the concept
3. use his/her personality and experiences to add context to the concept
4. record the information to transform it into a presentable format. Boiko uses the word *information* to refer to all common forms of recorded communication, including text, sound, images, video and animation, and computer files.

Information in turn can be further refined into *knowledge* by "the aggregation of disparate pieces of information, [and] the filtering out of irrelevant parts" (Hey 2004, 14). Hey (2004, 15) visualises this structuring or refinement process with a *knowledge pyramid* (shown in the figure below), where "large amounts of data are distilled to a smaller quantity of information, which is, in turn, aggregated to create yet more distilled, though more widely applicable, knowledge".

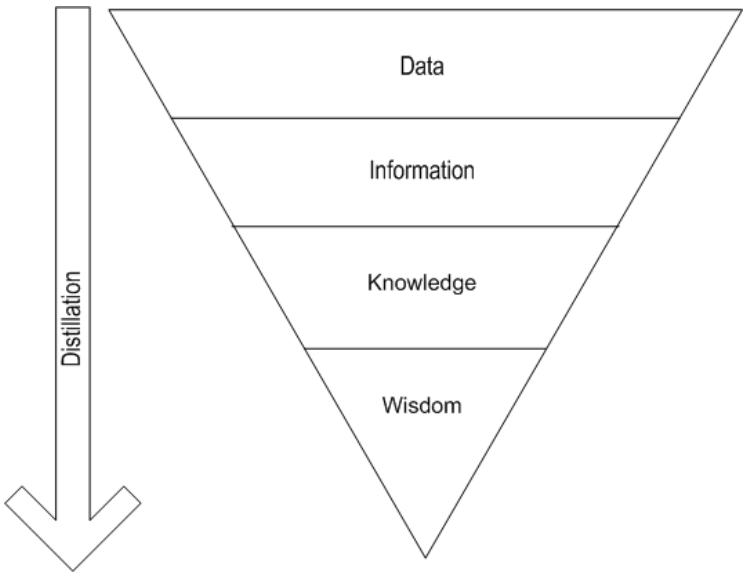

Figure 6. The knowledge pyramid (adapted from Hey 2006, 3)

Hey (2004, 6-9) describes some characteristics of data and information in terms of metaphorical analysis. Data and information are similar in the sense that both can be considered quantifiable, manipulable objects or resources. Data, on the other hand, is a solid, physical substance, while information may resemble a liquid, especially when there is more of it than we can handle. For example, information "*pouring* all over the Internet" can become *overwhelming*, turning into a "sea of information". Boiko (2005, 8) uses similar terms as Hey to describe information: it flows continuously and has no standard start, end, or attributes.

Miller (2002) and Wilson (2004) expound on the idea that data and information are manipulable objects. They argue that data and information can be captured, organised, documented, or managed while knowledge cannot. Furthermore, Miller and Wilson describe information as static and lifeless by nature: information has no intrinsic meaning while knowledge is the uniquely human ability of creating meaning from information in the mind and only in the mind. In other words, only a knowing individual can use the mental processes of understanding and learning to assimilate and incorporate information, thus turning it into knowledge and meaning. Although these mental processes normally also involve interaction with the world outside the mind, and interaction with others, no two individuals can have a similar knowledge structure in their mind. Moreover, as pointed out by Miller, "our interests, motivation[s], beliefs, attitudes, feelings, sence of relevance etc are always personal and [constantly] changing". Consequently, the meaning or knowledge built from messages (such as oral, written, graphic, or gestural messages) by a receiver can never be exactly the same as the intended meaning or knowledge base of their sender.

Knowledge is often divided into *tacit* and *implicit knowledge*. The word *tacit* means implied, indicated, or silent[1]. *Tacit knowledge* therefore means silent or hidden knowledge that is hidden even from the consciousness of the individual posssessing it, inexpressible, and may only be demonstrated through our acts (Wilson 2004). Wilson (2004) defines *implicit knowledge* as "that

1. "tacit." Merriam Webster Online 2005 (http://www.merriam-webster.com/)

which we take for granted in our actions, and which may be shared by others through common experience or culture". According to Wilson (2004), examples of implicit knowledge include mental models such as schemata, paradigms, perspectives, beliefs, and viewpoints.

Furthermore, both Miller (2002) and Wilson (2004) question the popular assumption that tacit knowledge can be captured and thus turned into expressible, implicit knowledge, which, when expressed, becomes information. They argue that this is due to the misinterpretation of Michael Polanyi's work *The Tacit Dimension* (1966). I agree with Miller and Wilson and, to align this study with their argumentation, have replaced the word *knowledge* with *information* whenever an author talks about concepts such as *knowledge management* or *knowledge capture*. I have marked this using square brackets ([information]).

2.2 From information to content management

What, then, is the relationship between *information* and *content*? Looking at dictionaries, *content* is defined as (my italics):

- "the *topics* or *matter* treated in a *written work"* or "the principal *substance* (as *written matter*, *illustrations*, or *music*) offered by a World Wide Web site" [1]
- "the *ideas*, *facts*, or *opinions* that are contained in a *speech*, *piece of writing*, *film*, *programme* etc" or "the *information* contained in a *website*, considered separately from the software that makes the website work" [2].

Based on these definitions, it seems that *content* refers to *information* that is *contained* in some kind of *medium*. This definition, however, is not exhaustive enough for the purposes of this study. As my aim is to create an information model that can be used as a framework in the OSS content management process, I will compare the concepts of *information management* and *content management*

1. "content." [4, noun] Def. 1b, 1c. Merriam Webster Online 2005 (http://www.merriam-webster.com/)
2. "content." [1, noun] Def. 3, 4. Longman Dictionary of Contemporary English Online 2008 (http://www.ldoceonline.com/).

to better be able to define what is the essential difference between the concepts of *information* and *content* for the purposes of this study.

Let us first have a closer look at information management. Robertson (2005) sees it as an umbrella term that encompasses all the systems, processes, and practises related to the creation and use of information within an organisation. Information management also deals with information itself, that is, the structure of information, metadata, content quality, among other things. Thus, information management encompasses the people, processes, technologies, and content used within an organisation. It follows from this that content management is one of the many facets of information management. Information management encompasses technologies such as content management systems (CMS), document management systems (DM), library management systems (LMS), and software configuration management (SCM) (Robertson 2004a; CM3 2008).

Information management is sometimes confused with the term *knowledge management*. However, as was explained in section *2.1 From data to information and knowledge* on page 15, the concept of knowledge management is an impossibility. For example, Wilson (2002) notes that (my italics)

> *knowledge management* is an umbrella term for a variety of organi[s]ational activities, none of which are concerned with the management of knowledge. Those activities that are not concerned with the *management of information* are concerned with the *management of work practices*, in the expectation that changes in such areas as communication practice will enable information sharing.

I have therefore used *information management* instead of *knowledge management* when the latter term occurs in the literature that I am using as theoretical background with the meaning stated by Wilson above.

What, then, is a content management system? Two persons, CMS consultants even, can rarely agree on the meaning of this term (Boiko 2007, 65; CM3 2008). CM emerged as a way to manage large web sites, but its role in an organisation is broadening. At the same time, there are no universally accepted standards about what content management systems are or do. (Boiko 2005, 66, 82) In this study, I will use the definitions of *content* and *content management* provided by Boiko

(2005) as he describes these concepts quite thoroughly: Boiko's definitions are in line with those of Hackos (2002), although they use somewhat different terminology.

According to Boiko (2005, 8-9), a piece of *information* can be transformed into *content* if it is given a usable form, intended for one or more purposes. This transformation process has a specific purpose in our information age: instead of reducing information to mere *data*, we can capture whole, meaningful chunks of information, and wrap them into descriptive *metadata*; a simplified version of the context and meaning of the original pieces of information. In other words, this tagging of information with metadata is an attempt to decrease its haziness and ambiguity and to make explicit the context, connotation, and interpretation originally meant by its composer. (Boiko 2005, 11) It is the metadata that allows content management, that is, the use of computer systems to collect, read, manage, process, and publish chunks of information: "If content management is the art of naming information [...], metadata is the set of names. In other words, content management is all about metadata." (Boiko 2005, 497) Boiko (2005, 492-493) defines metadata as "a set of standards that groups agree to for information definitions". The creation of metadata standards is extremely important as the standards form the basis of any kind of data sharing (for example, sharing data across applications). Furthermore, they can also bring large-scale efficiencies in information interchange among distributed groups of people that may not even know one another. If all the people within an organisation or community follow the same metadata standards, everyone can automatically reuse the efforts of one individual or group. This is a very important observation given that one of the purposes of this study is to find ways to make content creation, sharing, and reuse more open both within and between OSS communities.

To conclude the discussion, we can define content as:

- Rich information that is named, i.e. wrapped in simple metadata to compromise between the usefulness of data and the richness of information (Boiko 2005, 12).
- Information and functionality that has been captured, structured, and organised around a specific purpose, to be put into some particular use (Boiko 2005, xv).

Consequently, content management can be understood as:

- The art of giving names (in the form of metadata) to pieces of information. These names provide simple and memorable containers in which to collect and unify otherwise disparate pieces of information, and, furthermost, help datatise information to a certain extent. (Boiko 2005, 47)
- An attempt to gain control over the creation and distribution of information and functionality (Boiko 2005, 65).
- A process of collecting, managing, and publishing information to whatever medium (Boiko 2005, xv).

2.3 Comparing the openness of code and content

To summarise the discussion in the previous sections and to establish the answers to the questions stated at the beginning of this chapter, I will discuss *openness* in relation to the so-called DIKW (Data, Information, Knowledge, Wisdom) transition process (Clark 2004; Hey 2004), which is depicted in the continuum of understanding (see Figure 7. below) presented by Clark (2004). Furthermore, I will analyse the role of content management (and thus the information model) in this transition process and estimate if and how content management can aid an OSP to transform information into organisation-wide knowledge and/or wisdom. Later in this study I will provide other ways to look at the openness of OSS code and documentation, but in this conceptual analysis I will estimate openness in terms of characteristics typically used to describe data and information, shown in Table 8.

Table 8. Characteristics of data and information

Open	Closed
Concrete	Abstract
Explicit	Implicit, tacit
Unambiguous	Ambiguous
Clear	Hazy

Table 8. Characteristics of data and information

Open	Closed
Reusable	Single use, expendable
Transferable	Nontransferable
Manageable, manipulable	Intractable

Clark (2004) has founded his representation of the continuum of understanding (see Figure 7. below) on an article by Harlan Cleveland (1982). Clark's argumentation is aligned with that of Hey, Miller, Wilson, and Boiko, presented in the earlier sections of this chapter. Clark also characterises information as static and knowledge as dynamic. Furthermore, he too stresses the role of context in the DIKW transition process.

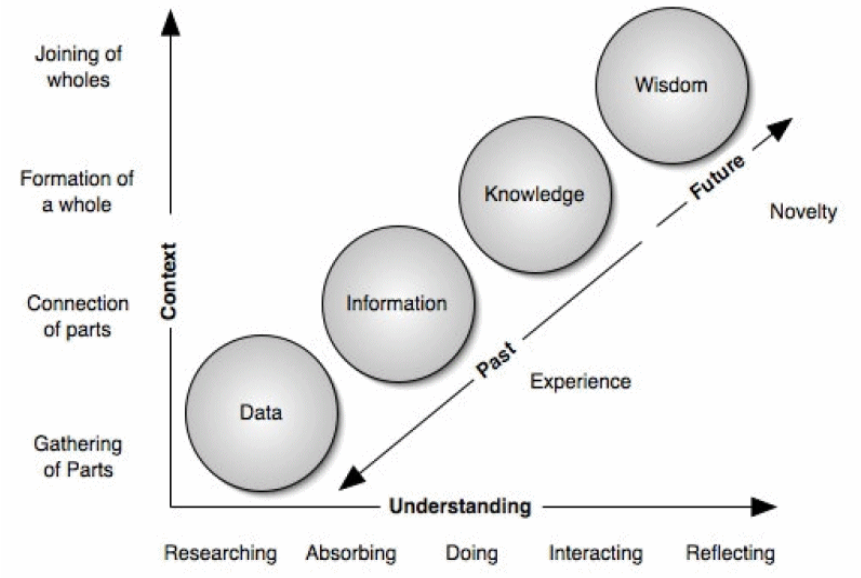

Figure 7. The continuum of understanding (Clark, 2004)

But Clark also adds some interesting new insight to the comparison of *data*, *information*, *knowledge*, and *wisdom*: he argues that *data* and *information* deal with the past, while *knowledge* deals with the present. Clark sees *wisdom* as the ultimate level of understanding: *wisdom* emerges from *knowledge* as we weave past experiences, that is, *context*, into new knowledge by absorbing, doing, interacting, and reflecting. Wisdom is what allows one to deal with the future, and to envision and design for what will be, rather than for what is or was. To be able to share wisdom, one must be able to

express ones personal experiences, the building blocks for wisdom, with a thorough understanding of the personal contexts of the intended audience.

It is, however, not only the goal of individuals to progress towards wisdom on the continuum of understanding: companies also attempt to employ methods such as "knowledge management" to ensure that wise decisions about future directions and designs can be made. Moving from information to knowledge and, ultimately, wisdom is particularly important for OSPs, which can be categorised as "people-oriented and knowledge-intensive software development environments" (Sowe et al. 2007, 2). However, as "knowledge management" was in a previous section found to be an impossibility, organisations should focus their attention on information management, content management being one of its many facets. The arguments presented by Maiju Markova (2005), who has studied the significance of knowledge for organisational change and renewal in a Knowledge-Intensive Service Organisation (KISO), also demonstrate the importance of the DIKW transition process for an OSP. In her work, Markova (2005, 9) has defined KISO as an enterprise or some other type of organisation that produces knowledge-intensive services that are to a large extent based on expertise and know-how. KISO is furthermost a socially constructed system that aims to produce *knowledge* and services to customers through interaction and problem-solving. The extensive amount of interaction required is the best exemplification of the complexity of such an organisation. In a KISO, knowledge is the most important resource required to produce services, but it may also represent the process or the end product itself (Markova 2005, 12). As I will show in Chapter 4, all of these characteristics most certainly hold true for OSPs. Furthermore, Markova (2005, 9) lists research and development organisations as one example of a KISO. Markova (2005, iv) argues that:

> The change and renewal of [a] KISO is highly dependent on how [information] in its different forms is used in internal processes of the organisation. In order for [a] KISO to change and renew holistically and efficiently, the organisation should recognise its own [information] needs, and balance both internal and external [information] exploitation and exploration. Furthermore, the change may be versatile in nature, e.g. incremental or radical change. The continuous use, sharing and development of organisational [information] have

> been noticed to generate incremental change, whereas the gathering and creating of entirely new [information] may generate radical change.

In her conclusions Markova (2005, 57-58) notes that the experts of the KISO and the balancing of information creation and exploration form the KISO's most valuable corporate asset:

- The existence of knowledgeable individuals is not sufficient by itself to ensure that the organisation will be wise: the know-how of individual employees (or in the case of an OSP, users and contributors) must be transformed into organisational knowledge for it to benefit the operations of the organisation as a whole.
- The knowledge of the individuals is *tacit knowledge*, acquired through experience. It is therefore extremely slow, if not outright impossible to transform it into organisational knowledge.
- The organisation needs to be able to identify how much and which parts of this knowledge could be utilized for the benefit of the entire organisation.

Close connections and cooperation with clients can have a substantial effect on the ability of the KISO to transform and renew itself. Close interaction can also help KISOs to develop and change in alignment with their clients. In the case of OSPs, however, we should talk about the OSS community instead of clients.

At this point it can be concluded that the conceptual analysis has proved the importance of the DIKW transition process for organisations such as KISOs or OSPs, but the tools and techniques of achieving this at an organisational level still resist clear definition. Miller (2002) calls this "the dilemma of our information age":

> Through technological innovation and breakthroughs in science, it became possible to deliver information (i.e. messages) accurately - and in an instant - to others, wherever they live on the face of the globe, whether we have any life experience in common with each other or not. **And therein lies the essence of our problem and the cause of so many of our quite tragic human and organi[s]ational dilemmas.** We can send information and provoke a response in almost anyone we wish anywhere on the planet, but we can never be sure - unless we know these people personally - how they are likely to interpret (i.e. what meaning they are likely to make of) the information they receive from us.

Furthermore, this is also the dilemma of all communication and documentation, OSPs included, as expressed by Miller (my italics):

> [...] attempts to *capture* (i.e. make *explicit*) human intentions serves only to transform them into intrinsically meaningless symbols even if made efficiently accessible from procedure manuals, computer databases, intranets, and other sophisticated information sources. Captured information always relies on responsible people [...] interpreting it within a *context* - and sharing and comparing interpretations where alignment to business purpose is a desired outcome.

As we have seen, the same themes of capturing, structuring, and distilling information, and the addition of context are repeated throughout the discussion as the only available methods of transforming information into knowledge and ultimately to wisdom. Furthermore, as metadata can be defined as a simplified version of the context and meaning of the original piece of information, it can be concluded that content is the missing link for constructing a continuum of understanding at an organisational level. Furthermore, to paraphrase the words of Boiko (2007, 52), the information model of a CMS is in fact an attempt to model knowledge as it exists in the brain of an OSS community.

Miller's demand for responsible people interested in comparing interpretations of captured information is reflected in the recommendation by Hackos (2002, 132) that the information model "must be designed by those who take the time to study and understand the prospective users". For example, information models that are understandable to experienced individuals are often equally obscure to newcomers. Moreover, CMSs that are useful for information authors may not suit the end-users of the information if they do not understand the underlying information model(s) (Hackos 2002, 131-132).

Open (that is, transparent, understandable, and accessible) information models can provide a way to review, improve, unify, and standardise metadata usage across OSPs. This in turn can help OSPs turn the information and know-how that they possess into organisation-wide knowledge. While striving towards this goal it is important, however, to bear in mind the following word of caution by Morville and Rosenfeld (2007, 4):

> No document fully and accurately represents the intended meaning of its author. No label or definition totally captures the meaning of a document. And no two readers experience or understand a particular document or definition or label in quite the same way.

Finally, let us return to the original question of comparing the openness of open source code and documentation to understand if and how code is more open by nature than documentation. I will break down the concepts of open source code and documentation to the level of *data*, *information*, and *content*.

Firstly, the source code of a software can be considered to consist of *data* and *functionality*. Boiko (2005, 31-32) defines functionality as "a computer-based process". In software, functionality exists in small chunks known as objects. A user interface represents these functionalities as a set of buttons, menus, dialog boxes, and other controls. Furthermore, Boiko (2005, 35) also categorises functionality that has been "packaged for reuse in objects or in blocks of programming code" as content. In today's software development, data, functionality, and content intermingle and become hard to distinguish: programmers create and package functionality into programming objects and then glue them together into an application. Programmers who know how to access the functionality in an object created by someone else can easily include it in their own programs. (Boiko 2005, 32-33) Since much of the functionality of an application may come from outside the application itself, building complex software that "combines the best functionality and data from a variety of sources is becoming easier and faster than ever" (Boiko 2005, 32). This is especially true of OSS.

If a software developer adds comments to his/her code, then the source code can be considered to be a combination of *data*, *functionality*, and *information*. Nevertheless, for another developer with the required know-how, the source code is certainly less ambiguous than an attempt to describe the design and functionality in a manual or developer's guide. For example, a Chinese programmer may be able read the code written by an American programmer, although s/he may not be able to understand a manual that has been written in English. Furthermore, if we discuss the role of a programmer who is writing source code in terms of the continuum of

understanding, we can see that the programmer is using his/her knowledge and wisdom, and disintegrating it into pieces of discrete data and functionality (and information in case the code is commented). This view is validated by Ye and Kishida (2003):

> Software systems are cognitive artifacts whose creation is a process of knowledge construction that requires both creativity and a wide variety of knowledge about problem domains, logic, computer, and others. In this sense, software systems, like books, are a form of knowledge media. Many OSS systems come into existence as results of the learning efforts of their original developers who try to understand how to model, or to change, the world with computational systems [...]. When the source code become accessible to users, the knowledge and creativity therein also become accessible, providing the initial learning resource that attracts users to form a community of practice around the system. By participating in the community, developers and users learn from the system, from each other, and share their learning with each other [...].

A document, on the other hand, may consist of *information* or both *information* and *content*. Compare, for example, two documents, one of which has been written with a word processor such as Microsoft Word while the other has been written in XML. The document written with a word processor may or may not use templates or style tags. It may or may not be stored using metadata that can be used to locate it and allow other authors or readers to deduce its contents without opening it. On the other hand, the document written in XML may have been constructed from several individual XML files, each of which is identified with metadata to allow the assembly. Furthermore, ideally the XML tags used represent semantic metadata instead of formatting styles. The document written with XML may be automatically converted into a completely different format such as HTML before publishing. The document written in XML thus fulfills the requirements of *content* established in the previous section better than the document written with a word processor.

In the figure below, I have tried to present a continuum of openness. I have envisioned data and information to exist almost at the opposite ends of the continuum because information requires a web of unstated relationships (context) to become usable, but data is the most concrete form of communication as it is so raw and discrete that no conversation is necessary to interpret

or understand it (Boiko 2007, 49-51). "To possess a piece of data, you simply must remember it" (Boiko 2007, 54).

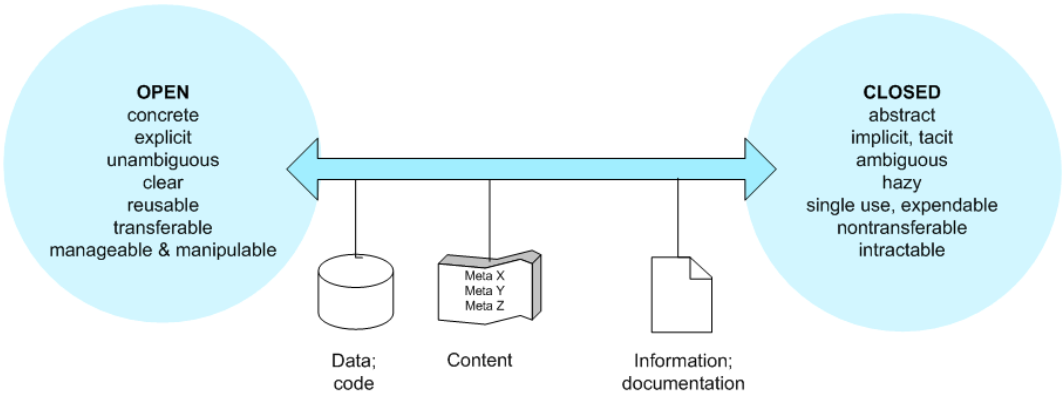

Figure 8. Openness of data, content, and information

Open source code and content, however, should be placed more approximately as programming objects (i.e. functionality) serve the same function as information topics and/or content units, that is, the basic building blocks of information identified in the information model. Both allow and provide a degree of separation between the person who creates them and the person who uses them. Furthermore, later a user of the object, topic, or content unit can find it and use it based on the attributes or metadata assigned by its creator. (Boiko 2005, 33) (Topics and content units will be described in detail in the next chapter.)

An information architect creating an open source information model moves in the same direction on the continuum of understanding as a programmer when s/he attempts to:

- model the knowledge structures that exist in the brain of both end-users and developers
- chunk information into manageable topics or content units
- datatise information by tagging it with metadata.

To conclude the conceptual analysis, open source code (or functionality) and open content need to share the following attributes:

- **segmenting and chunking**: both content and functionality can be divided into chunks as small as needed

- **sharing and reuse**: both content and functionality must be easy to locate and reuse apart from the application/publication for which it was originally intended. Using the content or functionality in your publication or application does not require that you know how it was created or how it works, you just need to know how to access/include/invoke it and what kind of results it can deliver. (Boiko 2005, 33)
- **design and modification**: I added this attribute to Boiko's list: if required, the underlying design must be available to allow fixing issues or improving the content or functionality.

3 Definition and structure of the information model

In the Introduction, information model was defined as a framework required to build a CMS that meets the community's needs. But information models also exist everywhere in our everyday lives: we create and use them unconsciously every single day. Libraries, for instance, are a familiar example of an institution whose daily operations are firmly founded on information modeling and content management. When you go to your local library, every book and item in the library has its own place, and it is easy to find what you are looking for as you are well familiar with the library's filing and organising systems. But there are also numerous examples of smaller-scale information models to be found everywhere around us: take your favorite cookbook or newspaper, for example. Both the newspaper and cookbook are written and organised in a manner that allow you to jump right on to the sports page, if you happen to be a sports fan, or quickly find your favourite recipe for that delicious chocolate cake. The articles in the newspaper are likely to be organised under main categories such as Foreign Affairs, Business, Sports, Entertainment, Weather, and so on. The recipes in your cookbook might be categorised according to the role in the meal (soups, salads, main dishes, desserts etc) or ethnicity (French, Italian, Mexican, Indian etc). Futhermore, if someone asked you to compose a short article for a journal about your field of expertise or to write down your great-grandmother's famous recipy for apple-pie, you would consider this a quite straightforward task, as you are already familiar with the *information types* of newspaper article and recipe, as well as the *content units* used to construct these information types. The end result is likely to resemble the outline shown in the following table:

Table 9. Familiar examples of information types and content units

Information type	Recipe	Newspaper article
Content units	Name	Headline
	Ingredients	Subtitle
	How to prepare (step-by-step procedure)	Standfirst
	Preparation time	Body text

Hackos (2002, 123-124; 136) defines the information model as an organisational framework that allows the information resources of an organisation to be:

1. categorised,
2. named or labelled,
3. organised and structured,
4. delivered and reused in a variety of innovative ways, and
5. effectively searched and retrieved by both users and authors.

Consequently, an information model must represent the points of view of both the authoring and user community. To achieve this, the categories defined in the information model must emerge from an analysis of the author and audience requirements (Hackos 2002, 132). Thus the OSS community forms the foundation of OSP information model. As shown in the figure below, the better the community's needs are reflected in the information model, the better the information model will be. (Hackos 2002, 9-10)

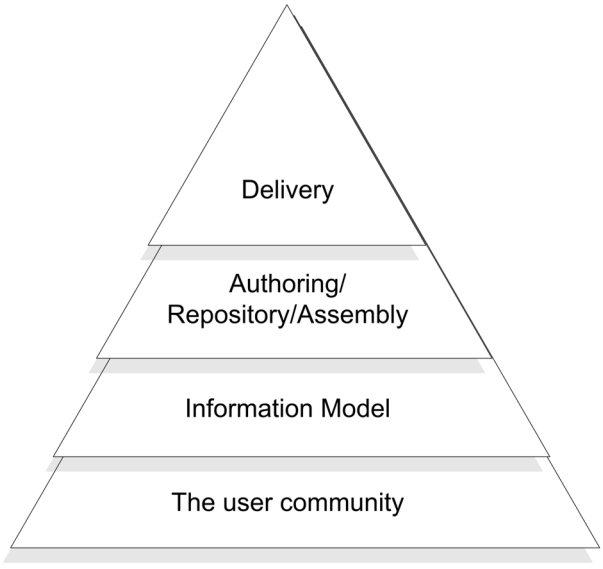

Figure 9. Conceptual model of a content management solution (Hackos 2002, 10)

The information model provides the names (i.e. the labels, metadata, terminology, or taxonomy) used to identify all the elements in the content repository (Hackos 2002, 40). Furthermore, it defines the organising and structuring principles behind the navigation design for all publication

media (Hackos 2002, 43-44). Lastly, it guides the choice of the technology best meeting the project's needs (Hackos 2002, 39).

The information model has a three-tiered structure:

- *dimensions of use* or *user-oriented metadata dimensions* define the attributes and values used to label the modules of content (derived from the needs of the user and author communities)
- *information types* provide authors with the basis for creating well-structured modules or topics that represent a particular purpose in communicating information (derived from the nature of the information itself)
- *content units* describe the chunks of content that are used to construct each information type. (Hackos 2002, 126)

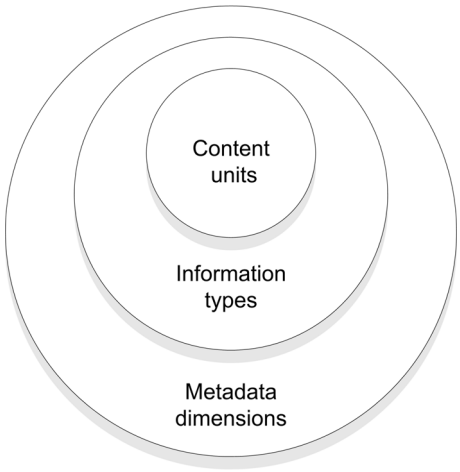

Figure 10.The three-tiered structure of an information model (Hackos 2002, 126)

The three-tiered information model (shown in Figure 10.) also reflects the definition of *content* that was established earlier in Chapter 2: *information* can be transformed into *content* by wrapping it in metadata. Moreover, information types are in effect just another form of metadata. The following sections describe information types and content units in more detail.

3.1 Information types

Hackos (2002, 161-162) defines information types as "subject-matter-related categories of information that authors use to create a consistent, well-structured topic". A topic is any stand-alone chunk of information that does not require another topic to be understood and it can be of any size. As topics are the key to content management and web delivery, information types are a central dimension of every information model.

Every topic of information included in the CMS must be assigned an information type and labelled accordingly. Information types can be strictly defined by creating templates or loosely defined by creating guidelines on how to write a specific type of topic. Ideally, there should be a unique template corresponding to every information type included in the information model to ensure that authors write consistently. (Hackos, 2002, 164) Structured information types (e.g. use of XML) can assist authors in the following ways:

- all authors include the same content within each topic
- authors can be confident that they have included the required content and excluded irrelevant information
- authors (both experienced and inexperienced) can write more quickly
- new authors know what is expected of them
- it is easier for editors and reviewers to determine what is complete and correct
- authors are able to find reusable information modules, which eliminates the need to rewrite, edit, or revise information
- information typing enables single-sourcing. (Hackos 2002, 178-179)

But information types not only assist authors, they also ensure that the content, structure, and organisation is consistent and thus provide "a consistent look and feel to the information". Consequently, they also assist readers in locating and understanding the information. (Hackos 2002, 165)

There are some information types such as procedures, concepts, warnings, specifications, and tutorials typically identified within the field of technical communication. But establishing standard information types for technical communication is quite challenging because technical information is so much more diverse in nature in comparison with the examples given at the beginning of this chapter. Furthermore, as Hackos points out, the field of technical communication has "fewer traditions to govern the selection and development of information types". (Hackos 2002, 181) Hackos (2002, 187) recommends that the information architect begins the task of identifying information types with the requirements of the user community rather than existing legacy information. She also notes that in most cases, the information types need to be unique to match the business and products of the company or organisation. Furthermore, to keep things simple and thus more manageable, the information architect should begin by identifying a minimal set of information types and add new ones as the need for them emerges. (Hackos 2002, 192)

3.2 Content units

The information architect also needs to define a semantic map that shows what components, known as *content units*, an information type contains, in what order they should appear, and which content units may be optional (Hackos 2002, 199). Content units are the smallest chunks of information identified in the information model. They are also the basic building blocks of information types. Some content units are unique to an information type, while others are common across information types within an organisation; some content units may even be common across an industry. (Hackos 2002, 168)

Hackos (2002, 203) recommends that "content units, like information types, should be defined organically, depending upon the needs of the users of information and of the subject matter itself". It is difficult to find or establish standard content units. Furthermore, just as with information types, the semantic map of content units should be determined based on an analysis of the users' needs, rather than deriving it from existing legacy information. (Hackos 2002, 205)

The concept of semantic map reflects Hackos' (2002, 208-209) recommendation that the content units should, where relevant, be tagged with semantic metadata to identify the meaning (e.g. *warning*, *task title* or *tip*) and not the format (e.g. style tags such as *heading* or *paragraph*) of the component. This is also in line with the definition of metadata established in Chapter 2.

4 Assessing the information needs related to open source

As has been pointed out by researchers, many definitions about OSS exist (see e.g. Ye and Kishida 2003). Some definitions focus on the aspects of the business model or licencing, while others tend to stress the ideological aspects of OSS development. Rothfuss (2002, 80) argues that "Several authors have developed theories to explain the [o]pen [s]ource phenomenon. Each of these theories contributes to the general understanding about [o]pen [s]ource, but all theories are incomplete and focus only on selected aspects." Rothfuss (2002, 87-88; 101) goes as far as to question the attainability of establishing a definitive theory of open source and suggests that scholars and researchers should instead aim to collect existing data and information about open source to build an open source framework that might offer new insights, useful predictions, practical advice, and good tips on the successful application of open source.

Consequently, this chapter does not aim to give an exhaustive theory or narrative of the open source phenomenon and its history. Rather, I will describe those aspects of OSS that are relevant for what Hackos (2002, 36) calls the information needs assessment phase of building an information model and which therefore also lay the foundation for building an open source content management framework.

According to Hackos (2002, 37), the needs assessment phase must address the following fundamental questions:

- Your users need information — what are their needs?
- You have information resources in your organi[s]ation — what are they and how are they produced today?
- You have processes used by people who author content even if those processes are not formally defined — how and how well do these processes work today?
- You have technology in place — how effectively is it being used and how adequate is it to the task ahead?

I will aim to answer these questions in the following sections, but will begin by discussing the characteristics of open source itself, in order to be able to relate the requirements of "openness" to the requirements of authoring and delivering "open content".

4.1 Definitions and characteristics of open source

In the previous chapter, "openness" of OSS code, documentation, and content were compared using conceptual analysis. In this section, I will discuss some existing definitions of open source to show which aspects OSS researchers consider as prerequisites for the "openness" of open source in order to later compare whether these aspects are also present in the environment where OSS-related content is being authored, and, if not, how could or should they be taken into account in the information model.

The official definition of open source (given below) is controlled by the The Open Source Initiative (2006), a non-profit corporation formed to educate about and advocate for the benefits of open source. To be considered open-source, a software must comply with the following criteria:

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

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

OSI has also trademarked the term *open source* to protect its meaning (Mork 2006, 23). Of the ten criteria listed above, especially important are the first two stating that the license must not require any fee for the distribution of the software and that the source code must be included in a well-readable form together with the program (Stürmer 2005, 15; Mork 2006, 23). In addition, Rothfuss (2002, 21) explains the rationale behind the third criteria, *derived works*, by noting that the ability to read the source code is not enough to support sufficient code review and rapid evolution but that developers must be able to experiment with the code and redistribute the modifications they have made to it.

Scholars and researchers also provide numerous different views on what the word *open* in open source really means or should mean. Bleek and Finck (2004, 10) argue that a software development process must fulfill the following three characteristics to be considered open source:

- **Openness**: The project must be open to new participants, e.g. new developers can get involved. Furthermore, openness means that anybody can use the product by simply installing it. It also means that the process itself is open to changes, which leads to the next point:
- **Agility**: The development process itself must be agile in the sense that the process is carried out in short cycles and may be changed as needed.
- **Distributed**: The participants of the development process are not all located at the same place.

Ye and Kishida (2003) provide more insight into the requirement of openness by arguing that the level of openness may vary. According to Ye and Kishida, "the different control structure inherent in each OSS community due to considerations in system quality creates different degrees of

openness that allows the legitimate participation and access of community members". Ye and Kishida discuss the possible combinations of openness using the following two dimensions: *product* and *process*.

In the *product* dimension:

- *open release* means that only formally released versions are accessible to all community members
- *open development* means that all interim development versions are accessible.

In the *process* dimension:

- *closed process* means that the discussion of system development is conducted mostly within an "inner circle", for example, through a strictly controlled mailing list that is not accessible to other community members
- *transparent process* means that although only the "inner circle" is involved in the development process, their discussion can be read by other community members
- *open process* means that development decisions are conducted in the public space: this allows the participation and access of all interested parties.

Stürmer (2005, 13) has interviewed eight active open source developers in his study. All of Stürmer's (2005, 36-38) interviewees also stressed the importance of "openness" although they expressed it in different ways. Stürmer's interviewees pointed out that an open source project should not only be "open to join", but also "open to leave". Another important aspect of openness was that the project should be "open to the choice of work", in other words, OSP leaders or core members cannot force contributors to do certain tasks, but everyone should have the freedom to choose a task that suits his/her interests and expertise. The interviewees also noted that open communication is a prerequisite of open source community building.

These various characteristics of "openness" are accomplished in a number of different ways in OSPs. As some of the definitions described above already imply, some aspects of openness are

dependent on the development process or on OSP infrastructure, while others may in addition require the use of particular tools. In addition to charting the information needs, the following sections also discuss these different "enablers" of openness.

4.2 Goals and typologies of an Open Source Project (OSP)

According to Hackos (2002, 37), the needs assessment must take into account "the point of view of each relevant group in the user community as well as that of the [content] authors". However, in the case of OSS, both the concept of community and the relevant groups must first be established before the corresponding information needs can be analysed. Moreover, identifying the relevant groups can be quite difficult as open source projects tend to be extremely versatile in nature, ranging from solutions targeted at a very specific, extremely technical audience (e.g. Cocoon[1]), to applications which tend to attract a wider, less technical audience (e.g. Mozilla Firefox[2] or the OpenOffice Suite[3]).

According to Stürmer (2005, 20), all contributors of an OSP together form the community of this project. He adds that

> In contrast to the single contributor's view where e.g. the motivation of the individual [is] important, the community view is concerned with topics like shared values and common vision. Thus the contributors' perspective is a look from inside the OSP whereas the community perspective is an outside view of the OSP perceiving the members of the project as a closed unity.

While most OSPs can be categorised as being "hybrid by nature, consisting of actors with both commercial and non-commercial interests, motivations[,] and backgrounds" (Vainio et al 2006, 4), they all have the same eventual goals. All OSPs aim to increase their community size, improve internal collaboration, and to further develop their software (Stürmer 2005, 9). Rothfuss (2002, 103) argues in the same spirit that "[t]he aim of each OSP should [...] be to do everything possible

1. http://cocoon.apache.org/
2. http://www.mozilla.org/
3. http://www.openoffice.org/

to ensure key contributors find a supportive environment, and to nurture participants to make them (future) key contributors".

Consequently, the information model must take into account the information needs of the OSS community as a whole in addition to those of the individual actors and groups within the community. The purpose of an open source information model should be to support an OSP in reaching the ultimate goals of community building and software development. To achieve this purpose I will first discuss different community typologies and infrastructures and later in this chapter relate the groups, actors, and their information needs to these different categorisations.

4.2.1 Categorising OSPs according to life cycle, stages, and typology

The typical life cycle of a successful OSP can be divided into the following phases (Rothfuss 2002, 38; Stürmer 2005, 15-16):

1. The OSP initiator comes up with an innovation for his/her own personal problem. S/he asks other people (for example friends and colleagues) for some feedback and initial support.
2. The interested persons start to exchange their knowledge on the topic and, as a result thereof, create a vague picture about the central issue at hand.
3. People who are interested enough and willing to spend some resources on the problem create an informal project and work on the issue until they find some satisfactory solution.
4. The project is considered to have started once the first source code commit has been made into a repository (Stürmer 2005, 15). Once the source code is available, an announcement is made using various news channels and open source platforms to attract potential users and contributors.
5. If the OSP is successful, the community members find the project attractive enough to become involved in it. Typically, they first try using the software, then suggest some improvements and may later on even start contributing themselves.

6. The community grows and evolves by adapting to the changing environment. The received feedback, new information and resources also allow the solution to grow and address the important issues more efficiently, using better strategies.
7. As the technical size of the project grows (measured, for example, in lines of code), the community also evolves by splitting into subprojects and by assigning new tasks to the community members according to skill, experience, and current project responsibilities.
8. The OSP community has become established. The OSP research cycle has turned full circle and begins afresh from stage three.

Rothfuss (2002, 38-39) and Stürmer (2005, 16-17) also relate the OSP life cycle to different stages of an OSP. The common classifications of an OSP according to the stage it is in as defined by Rothfuss are shown in Table 10.:

Table 10.Stages of an OSP (Rothfuss 2002, 38-39)

OSP stage	Description
Planning	No code has been written, the project scope is in flux. The project is at the stage of an idea. The project enters the next stage when tangible results appear in the form of source code.
Pre-alpha	Very preliminary source code has been released. The code is not expected to compile, or even run. It is very difficult for observers outside the project to understand the function/meaning of the source code. The project enters the next stage when a coherent intent is visible in the code, which indicates the eventual direction of the OSP.
Alpha	The released code is functional at least some of the time, and the solution begins to take shape. Preliminary development notes may show up. Active work to expand the feature set of the application continues. The project enters the next stage as the amount of new features decreases.
Beta	The code is feature-complete, but retains faults. These are gradually weeded out, leading to more reliable software. If the number and/or severity of faults is deemed low enough, the project releases a stable version, and enters the next stage.

Table 10.Stages of an OSP (Rothfuss 2002, 38-39)

OSP stage	Description
Stable	The software is useful and reliable enough for production/daily use. Changes are applied very carefully, and the intent of changes is to increase stability, not new features or functionality. If no significant changes occur over an extended period of time and only minor issues remain open, the project enters the next stage.
Mature	There is little or no ongoing new development as the software fulfills its purpose very reliably. Changes are applied with extreme consideration and caution, if at all. A project may remain in this final stage for many years before it slowly fades into the background because it has become obsolete, or has been replaced by better software. The source code for mature projects remains available indefinitely and may serve educational purposes.

Fogel (2005, 15) defines the alpha and beta stages slightly differently. According to Fogel, alpha refers to the first release that has all the intended functionality. While it still retains defects, users are able to use it for the intended purpose. In a beta release all the serious defects have been fixed, but the software has not yet been thoroughly tested. If no serious bugs are found during testing, the beta version may become the official released software. In effect, however, the alpha and beta stages are "very much a matter of judgment". Rothfuss (2002, 39) adds that an OSP can be considered successful if it is able to enter *stable* stage. On the other hand, he also notes that some projects remain reluctant to label themselves as being *stable* or *mature* because of perfectionism or a goal of "getting it right".

Consequently, the ultimate goal of an OSP is to evolve and renew itself quite dramatically until, or rather, *if* it reaches maturity. According to an estimate presented by Fogel (2005, 1) approximately 90-95% of OSPs fail. The OSP categorisations by Rothfuss and Stürmer explained above mainly relate to the state of the source code but provide little explanation about the factors that enable the code to evolve. For example, Mikkonen et al (2006, 62) note that "[a] company participating in a [...] open source software community needs a detailed understanding of different governance models, motivation structures, sizes, traditions[,] and ideologies of communities."

Consequently, it is relevant to look at other ways of analysing and categorising OSPs, such as community infrastructure.

In addition to the OSP stages, OSPs can also be classified according to variables such as:

- age (how long the OSP has been active)
- size (the amount of developers)
- maturity
- level of hybridity (the mix of volunteers and company employed developers)
- community ideology (shared values and goals)
- anarchy/hierarchy (the intensity and mode of organisation). (Vainio and Vadén 2006, 15)

Maturity can be seen as, for example, a derivative of community size and age (Vainio and Vadén 2006, 15), or the strength of the social and cultural ties, traditions, and common guidelines and practices (Mikkonen et al 2006, 63).

In their study, Mikkonen et al (2006, 62) identified two distinct types of community ideology and work ethic: a volunteer community and a company-based community. Furthermore, their categorisation coincides with that presented by Robertson (2004b) who concentrates more on the OSP's level of hybridity or business model, and distinguishes between community-based open source systems and commercially-supported open source systems. To combine the views of Mikkonen et al (2006) and Robertson (2004b), open source can be divided into:

- **Community or volunteer-based open source**: The "hacker work ethic", i.e. the traditional FOSS work ethic of freedom, fun and sharing of information, is dominant. The software is completely free, produced by volunteers across the world. While there may be a coordinating body, there is no owner of these systems in the conventional legal sense.
- **Company-based i.e. commercially-supported open source**: Recently, a number of commercial organisations have developed products based on the OSS development model. Here, there is one organisation that provides professional services or support for the product, while the code itself is freely available i.e. open source. Company and business

objectives naturally have more importance than in community-based open source. A large percentage of developers are paid for their contributions.

Robertson (2004b) notes that "in practice, many approaches are a mix of these two models".

In this section, different categorisations of OSPs or open source communities have been given. These categorisations represent the outside view of an OSP, while the following section provides an inside view of an OSS community and describes the groups and actors both within and without a community.

4.2.2 OSS community infrastructure, actors and roles

For Ye and Kishida (2003), the fundamental difference between OSPs and most closed source software projects is the *role transformation* of the people involved in an OSP. In closed source software projects, developers and users are clearly defined and strictly separated. In OSS projects, there is no clear distinction between developers and users: all users are potential developers.

OSS community roles are typically depicted using an onion model (Ye and Kishida 2003; Vainio and Vadén 2006, 13-14; Helander and Laine 2006, 51; Stürmer 2005, 14). The model defined by Ye and Kishida (2003), shown in Figure 11. below, is one example of such a model.

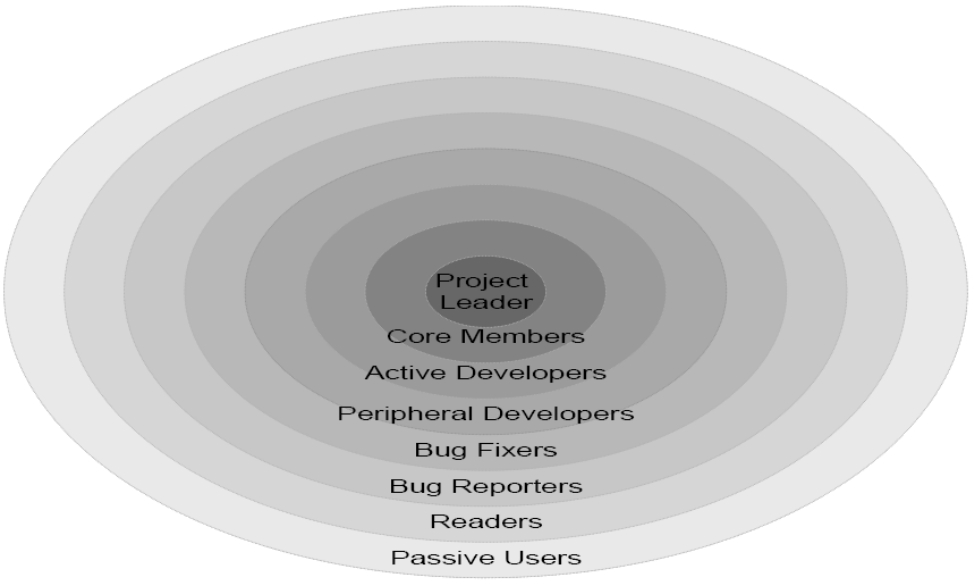

Figure 11. General structure of an OSS community (Ye and Kishida 2003)

The open source participants shown in Figure 11. can be described as follows:

- At the heart of the community is the *project leader*. S/he is often the person who initiated the OSSs project, and is responsible for its vision and overall direction.
- There is also often a group of *core members* or *core developers* that guide and coordinate the project. The core developers have been involved with the project for a relatively long time and have made significant contributions to its development and evolution.
- *Active developers* regularly contribute new features and fix bugs, and are thus one of the major development forces of OSS systems.
- *Peripheral developers* make occasional contributions, and thus their period of involvement is short and sporadic.
- *Bug fixers* fix bugs discovered by themselves or reported by other members of the OSS community. They must be able to read and understand a small portion of the source code of the system where the bug occurs.
- *Bug reporters* discover and report bugs. They assume the same role as testers of the traditional software development model: they do not fix the bugs themselves, and may not read the source code, either. The existence of this layer is important to assure the high quality of OSS.
- *Readers* are active users of the system. They try to understand how the software works by reading the source code, but do not contribute. Their motivations for reading can vary from a desire to learn good programming skills or to understand the OSS model/architecture to implement similar systems.
- *Passive users* use the OSS in the same way as most people use commercially available closed source software. While passive users have the least influence, they still play important roles in the community: according to Ye and Kishida their very existence "contributes socially and psychologically by attracting and motivating other, more active, members, to whom a large population of users is the utmost reward and flattery of their hard work". Ye and Kishida

also provide an interesting metaphor when they compare passive users to the audience in a theatrical performance who offer values, recognition, and applause to the efforts of actors.

The layered community model describes an OSS community in a number of different ways. The radius of the circle reflects the size of a given layer: the farther from the nucleus a layer is, the more participants normally belong to it. Furthermore, the darker the shade of a layer, the more involvement, responsibility, and power the group in question has in the community. Thus, the outer layer of the onion is the largest group and as one approaches the center, the group size decreases. (Helander and Laine 2006, 51) According to Vainio and Vadén (2006, 14), the so-called 20/80-rule (derived from the Pareto Principle), according to which 20% of the volunteers do 80% of the work, seems to apply to OSS communities as well. The onion model can also be seen as a ladder that the community members climb up as they want to become more involved in OSS development (Helander and Laine 2006, 51).

All of the above-mentioned roles may not always exist in all OSS communities: each community has a unique structure depending on the nature of the system and its member population. Furthermore, different communities may use different names for the roles defined above. It must also be noted that the roles are not static, and that the differences between them are usually very small. For example, a *passive user* may at one point become a *reader*, and a *bug reporter* may at some point try to fix a bug themselves and thus become a *bug fixer*. (Ye and Kishida, 2003) Most roles are typically enacted by several community members simultaneously (Rothfuss 2002, 58).

Ye and Kishida's model shown in Figure 11. can be criticised for prioritising the development of source code. For example, the terms *bug reporter* and *bug fixer* place the emphasis on *bugs* i.e. software defects, while tasks such as finding and fixing documentation defects should be considered as important. It would also be beneficial to better distinguish between *actors* and *roles*: in my opinion, the onion model represents *actors*, each of which may be playing many *roles*. Depending on the exact categorisation used, an *actor* such as a *core member* or *core developer* may

simultaneously be performing the *roles* of system administrator and programmer. In addition, depending on the value placed on non-coding tasks, an information architect responsible for creating and maintaining the project's information model might also be considered a core developer. Furthermore, a *reader* might be a technical writer interested in learning more about the design principles behind the project's content management solution and who therefore decides to have a look at the project's information model.

Another group of actors that is not included in the onion model is that of *project evaluators*, that is, people who have not yet downloaded or started using the software, but may be reading about it from the OSP's web site in order to find out whether they should. The content management system (and thus the information model) nevertheless needs to take into account the information needs of evaluators as well as those of passive users and contributors. The concept of *contributor* is also problematic: what does it really take to become a contributor? Stürmer (2005, 18) says that "[a]ll persons actively progressing the OSP are acting in the role of a contributor [...] [n]ot included are the inactive users who have downloaded and installed the software and might even have subscribed to the mailing list but never write a message nor actively show any other sign of their existence". Stürmer's view of the community and the different actors within the group of contributors is shown in Figure 12. On the other hand, if a project evaluator posts a question on the OSP's mailing list to ask if the software supports a specific environment, s/he may in fact be indicating a documentation defect and could thus be considered an issue reporter and contributor as the supported environments may not have been indicated clearly enough on the project's web site.

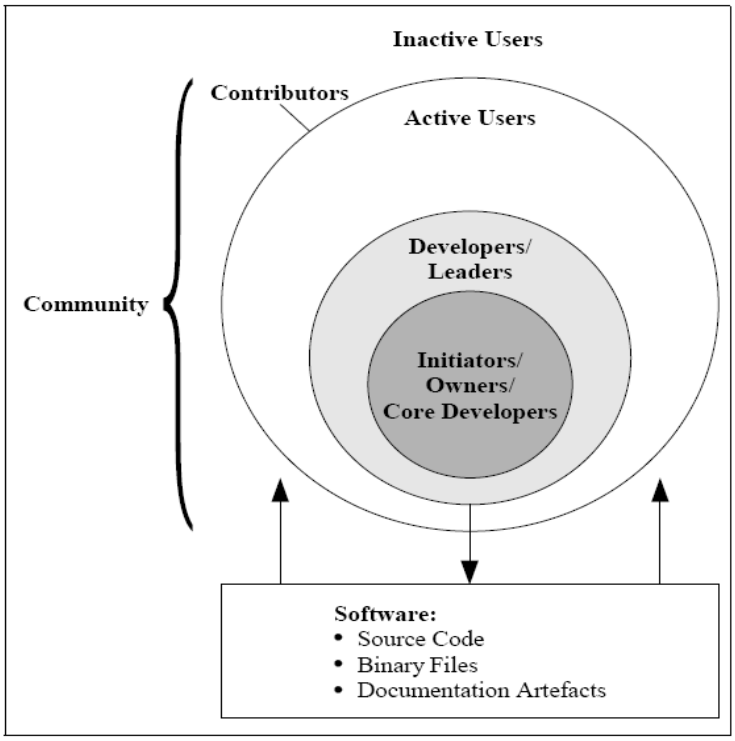

Figure 12.An abstract view of an Open Source Project (OSP) (Stürmer 2005, 14)

In this study, the word *contributor* has a broad meaning: a person may become a contributor just by posting a meaningful message on the project's mailing list. Thus, Figure 12. also applies for this study, but the line between an active user (i.e. a contributor) and an inactive user is very hazy. The term *developer* includes all kinds of development activities ranging from programming to documentation, graphics design, etc.

Nevertheless, the variety of actors and roles must be balanced for the community to be healthy. If there are no active core members, the project may lose focus. On the other hand, the community needs passive users as well as core developers, because a diversity of actors and roles ensures that the project is reviewed from all relevant perspectives. (Helander and Laine 2006, 51; Stürmer 2005, 27)

4.2.3 Profiling OSS developers and their motivations

OSS research already includes several empirical, typically web-based surveys of OSS users and contributors and their own understanding of the guiding motivations. In this study I will use the

FLOSS surveys and another smaller scale but more recent survey of four open source communities (Debian, Gnome, Eclipse and MySQL) by Mikkonen et al (2006) to construct an image of a typical open source developer.

The FLOSS surveys represent the largest and most comprehensive surveys of OSS developers worldwide. It was conducted in several parts:

- The first FLOSS survey was conducted in 2002 (appr. 2800 received responses)
- A FLOSS-US survey was conducted in 2003 (appr. 1500 received responses)
- Several FLOSS-JP/FLOSS-Asia surveys were conducted in 2003 and 2004 (appr. 650 responses). The first FLOSS-JP survey was conducted in English and because it received a quite small amount of responses, the researchers decided to implement two additional surveys (one in Japanese and one in the other Asian languages). (Mitsubishi Research Institute Inc. 2004; Ghosh 2004)

It should be noted that the roles of translator, technical writer and similar were not included in the survey done by Mikkonen et al (2006, 54). The FLOSS surveys did not explicitly define any criteria that the respondents should fulfill before they could take part in the survey. Instead, the FLOSS researchers simply asked everyone that considered him/herself to be an OSS developer to respond to the survey. (Mitsubishi Research Institute Inc. 2004)

All of these surveys provided similar results. A majority of the respondents were relatively young (in their twenties or thirties), well educated (for example, in the survey by Mikkonen et al 80 % of the respondents had completed an academic degree and 12,5 % had a PhD) and had a background in software related fields (e.g. software engineers or programmers). A great majority of the respondents were located in Europe and Northern America.

Only 1,5-2% of the FLOSS survey (Mitsubishi Research Institute Inc. 2004) respondents were female, while 4% of the respondents in the survey done by Mikkonen et al were female. I speculate that the slight increase in the amount of female respondents may relate to the fact that the study done by Mikkonen et al is more recent. But the marginal amount of female OSS devopers also

raises some other interesting questions. The small amount of women can be partly explained with the fact there has traditionally been fewer women than men within the field of computer science (Carlson 2006). However, there are certainly more women involved in IT today than just 2-4 %. I speculate that other possible explanations for the low percentages might include the low value placed on non-programming activities in OSPs. The title "OSS developer" is often considered to be rewarded only for persons who have made significant code contributions. Moreover, there are typically more women involved in fields such as technical communication and translation, and these roles certainly exist in OSS development in well. Is it so that the translators and technical writers who contribute to OSS do not consider themselves as developers, or are there extremely few professional technical writers or translators involved in OSS and if so, why? Are the barriers of entry too high for non-programmers, i.e. is open source not open for non-programmers? This calls for future research.

Unfortunately all of the surveys provide very marginal (if any) results about documentation-related activities in OSS. The survey questions of the last FLOSS surveys (FLOSS-US, FLOSS-JP, and FLOSS-Asia) were modified to also include documentation, but they still fail to provide enough useful information for the purposes of this study. While they included translation, localisation, and documentation as activities, the surveys do not, for example, define what is meant by *documentation*: i.e. does it refer to commenting code and/or writing user/developer documentation? On the other hand, the survey results seem to prove the theories that will be presented in the next section about how OSPs tend to focus on implementation and testing activities instead of documentation and design. (Mitsubishi Research Institute 2004)

According to the FLOSS surveys, approximately half of the developers received direct or indirect remuneration for developement activities. The percentages for direct monetary compensations were:

- 53,7% in FLOSS-Europe
- 43,2% in FLOSS-US

- 26,8% in FLOSS-JP
- 45,1% in FLOSS-Asia. (Mitsubishi Research Institute Inc. 2004)

These findings are very important because they indicate a need to investigate developer motivations: although the amount of paid OSS developers is probably increasing, it is more than likely that without motivated contributors there would be no OSPs. Therefore the motivations and aspirations of contributors are a central part of the use cases to be included in the information model. Both the FLOSS surveys and the one by Mikkonen et al clearly indicated that developers participate in OSS primarily to acquire and share information/knowledge. These results also validate the importance of the conceptual analysis performed in Chapter 2.

We currently know quite little about the content authors and their motivations (see e.g. Berdou 2007, 112). According to Hackos (2002, 38) during the needs assessment phase the information architect needs to "look closely at the processes that are used throughout your organi[s]ation to produce, approve, and disseminate information resources. Find out who authors, who reuses information coming from other parts of the organi[s]ation, and who reviews and approves information." At the moment there is a very limited amount of OSS research available that can be used in this step of the needs assessment phase. Consequently, a content-management-centered or documentation-centered OSS survey would provide valuable evidence for an open source content management framework. Berglund and Priestley (2001, 132) note that "[j]ust as open-source development blurs the line between user and developer, open-source documentation will blur the line between reader and writer. Someone who is a novice reader in one area may be an expert author in another." Furthermore, it is somewhat difficult to establish how active contributors and what types of contributions should be included in the survey; content contributions can be as small as posting a question on a mailing list. (The different forms of community authoring will be discussed in more detail in section 4.4.)

The most comprehensive research regarding documentation contributors comes from Oram (2007). After writing several articles about documentation contributors and speculating the

motivations behind contributions, Oram conducted a survey of his own. He received 354 responses, most of which came from OSS communities such as Perl, GNU, Linux, and GNOME. Oram stresses that personal connection strongly affected the response rates of the communities, and therefore no conclusions should be drawn from the pattern of responses from different projects.

Oram (2007) found that community building was the most popular motive for contributing documentation. Personal growth such as learning through teaching came second, mutual aid came third, and gratitude came fourth. Reputation building reached sixth or seventh place. Oram openly admits that he was surprised by the results and how the respondents felt that community building was more important than personal reputation building. Based on Oram's findings, it seems that Raymond's (2001) catch phrase "scratching your own itch" can also be extended to community authoring in a sense of mutual back-scratching: no matter what form it may be, each contribution reinforces "a culture of mutual aid" that indirectly produces eventual benefits for the contributors (Oram 2007). Based on the "burning desire to help others" that ran through the received free-form responses, Oram speculates that educating others is a "basic human imperative", essential for the survival of the human race from generation to generation. On the other hand, while reputation building did not make it to the top three, this does not mean it is unimportant: reputation building got 833 votes (75% of the most popular reason).

It was shown earlier that an OSS contributor may wear many hats, but it remains unknown to what extent active software developers also act as documentation developers in OSS, or if there are also technical writers involved, or whether it is the newbies who familiarise themselves with a project by writing documentation as was suggested in the Introduction. However, as I was searching for material for this study, I quite often came across comments where programmers lamented how difficult it typically is to find professional technical writers willing to volunteer in an OSP. I strongly suspect that the more traditional, in-depth documentation is typically written by software developers who also have a knack for writing. The results of Oram's (2007) survey might

also be seen to support this theory because the main motivation for contributing content was community building while the FLOSS surveys reported learning and information sharing as the most popular reason: if you are a professional in technical writing or information architecture, it would be likely that your main reasons for contributing content would be self-enriching motivations such as learning or monetary reward. Interestingly, this speculation leads to the same question already mentioned earlier in this section: are there extremely few professional technical writers or translators involved in OSS and if so, why? Again, more research is needed.

4.3 Software engineering and design in OSS

Significant differences in software engineering practices exist between different OSPs. Thus, it is not in the scope of this study to outline and/or examine the OSS development/engineering process in detail. As has been pointed out by Luoma (2006, 62), most OSPs seem to use parts of known software engineering processes in some form, while others may even use a full software engineering process. As it is impossible to tie the information needs of the OSS users and developers to a specific model of a software development process, I will only discuss OSS design, implementation and testing at a general level, mainly to explore the important differences compared to the development of traditional closed source software, and to be able to define some of the information needs related to these activities later in the information model (Chapters 5 and 6).

Typically, the software engineering practices of OSPs differ from those used in traditional closed source projects. In some cases, especially in volunteer-based communities, this may be a necessity and a result of the voluntary nature of the projects and the differences in motivational and social factors compared to closed source projects. Some of the differences may also be due to the resources available to OSS developers compared to the resources available to developers in traditional closed source projects. Consequently, it is possible that some practices used in

traditional closed source projects are simply not applicable to open source projects, and vice versa. (Luoma 2006, 62)

Luoma (2006, 58) defines software design as "an activity of software engineering, which concentrates on planning how to implement the features described in the software specification and requirements." According to Luoma, several levels of design are possible:

1. **Higher-level architectural design** aims at
    - "dividing the software into subsystems and components and defining the relationships between the components and subsystems"
    - producing "a design where changes inside an individual subsystem or component do not affect other subsystems and components", which is typically achieved through "modularity, strict interfaces, encapsulation, and applying known architectural styles and design patterns".
2. **Lower-level module design**
    - means the design of individual subsystems, components and modules
    - can also be further divided into several levels, as each subsystem may consist of further subsystems and components
    - the detail of lower-level module design can also vary between systems and processes used, but differences can also exist inside a single system, for example, when the critical parts are designed in more detail than non-critical parts
    - is often closely tied with implementation, that is, writing the actual code, and testing and thus done as part of the implementation phase.

OSPs typically focus more on implementation and less emphasis on the other phases of the software engineering process, such as system-level design, detailed design and support. This means in effect that the OSS design and architecture is thought to be either visible from the structure of the code, or to be found from the community discussion archives, or simply stored in the heads of the core developers. This holds true especially for detailed lower-level design. Some form of

higher-level design normally exists even for volunteer-based OSPs, but there are also exceptions to this rule. Moreover, in OSPs, the software design is typically documented *after* it has been implemented. Consequently, design documents are often outdated as the documentation is rarely arduously updated every time that changes to the corresponding code are made. (Vixie 1999; Luoma 2006, 62-63)

The implementation phase is central in OSS development simply because voluntary OSS developers find the implementation phase more interesting than writing formal design specifications. Field testing is also of increased importance in OSS development. Despite the lack of a systematic testing process, high quality is considered to be one of the major strengths of OSS. This is due to the relatively large amount of bug fixers and reporters (see Figure 11.), leading to massive parallel debugging and peer review. Young, small, or immature OSPs naturally do not benefit from this characteristic of OSS development in a similar scale as an established OSP does. (Vixie 1999, Luoma 2006, 62-66) Rothfuss (2002, 72) argues that quality assurance activities scale up better because they require less interpersonal communication and coordination than software design activities.

Despite the low emphasis placed on the design phase, software architecture plays an extremely important role in OSS as the nature of OSS development itself places a great deal of strain on the architecture. Modularity of the software is an extremely important prerequisite for OSS. Modularity means that the software is composed of components that interact with each other through defined interfaces but are otherwise independent of each other. Firstly, modularity allows developers to extend the original software in ways not foreseen at the beginning of the project. Secondly, it moderates the relatively steep learning curve of OSS: it allows enhancements and bug fixes to be divided into smaller tasks. Or, as put by Ye and Kishida (2003)

> To attract more users to become developers, the system architecture must be designed in a modulari[s]ed way to create many relatively independent tasks with progressive difficulty so that newcomers can start to participate peripherally and move on gradually to take charge of more difficult tasks.

It also enables developers to focus their efforts on modules and functionality that they are most experienced with and/or find most interesting. Consequently, modularity is an important "enabler" of openness. First of all, it enables agile OSS development. In addition, it helps make projects "open to join" and also "open to the choice of work". The positive impact modularity has on recruiting new developers, developer collaboration, and parallel software development is also demonstrated by Stürmer's (2005, 57-60) study.

The principle of software modularity is closely interrelated with the Application Programming Interface (API). According to Stürmer (2005, 23) the Application Programming Interface (API) "provides a set of routines, protocols and tools for constructing software applications" and "defines what routines are available, how to call them, and what they do, but it does not explain how the subroutines are implemented and what algorithms are applied". In other words, the API tells software developers "how to access e.g. core functionality of a software framework" so that they are able to write applications of their own. As Stürmer (2005, 57) explains, "[w]ell defined interfaces of the software parts are necessary to ensure smooth interaction of the contributions of all the developers".

While the discussion above showed that especially documenting the lower-level design is often overlooked in OSPs because the design is thought to be visible from the code, Mork (2006, 41) argues that such practices are insufficient when "used in narrow, speciali[s]ed domains where there are fewer interested developers". Moreover, several of Stürmer's (2005, 66-67; 72) interviewees stress the importance of API documentation including in-line comments within the code, reference manuals built using JavaDoc or PHPdoc, and also developer's guides. The interviewees also name one difficulty related to APIs: it can be difficult to distinguish between the internal and external API of an OSS meaning that one developer may continue to build upon an interface that was originally thought to be for internal use only (Stürmer 2005, 71). The internal and external APIs should therefore be defined in clear, written API policies.

4.4 OSS information resources and community authoring

In this section, I will align the OSS content management environment, process, and practices with software design and development, discussed in the previous section. To follow Hackos' (2002, 38) guidelines , I will perform "a complete inventory" of the OSS information resources and study the processes used to produce the information.

I will focus on *content management* instead of *documentation development* because, as was mentioned in the previous section, OSPs in many cases do not have extensive, formal documention and especially design documentation is often lacking. However, this does not mean that there is no information available about open source solutions. Rather, the information is recorded, collected, and distributed using different tools and media instead of traditional documentation. It is therefore important to find ways of integrating content management into these media in order to turn information into relevant, meaningful, and accessible content. This idea is also demonstrated in the following quotation taken from Fogel (2005, 28):

> Since almost all communication in open source projects happens in writing, elaborate systems have evolved for routing and labeling data appropriately; for minimizing repetitions so as to avoid spurious divergences; for storing and retrieving data; for correcting bad or obsolete information; and for associating disparate bits of information with each other as new connections are observed. [...] As much as possible, the communications media themselves should do the routing, labeling, and recording, and should make the information available to humans in the most convenient way possible. In practice, of course, humans will still need to intervene at many points in the process, and it's important that the software make such interventions convenient too. But in general, if the humans take care to label and route information accurately on its first entry into the system, then the software should be configured to make as much use of that metadata as possible.

During the writing of this thesis, it became more and more apparent to me that the relevance of the notion *OSS documentation* itself should be redefined, perhaps even questioned, as was already mentioned in the Introduction. Fogel's observations describe my findings perfectly: instead of trying to create and manage traditional documents, we should focus on identifying those information flows that need to be labeled and routed using content management to make sure that information will be accessible to those participants who need it. To support my argumentation, I

will first investigate how open source itself sees the concept of *documentation*. One OSP that is often noted for its high-quality documentation is the Linux Information Project (2006). Here is how they define documentation:

> Documentation is any communicable material that is used to describe, explain or instruct regarding some attributes of an object, system or procedure, such as its parts, assembly, installation, maintenance and use". [...] Documentation can mean different things in different contexts. [...] And documentation can take many different forms even in a single context.

This definition of documentation resembles the definition of information presented in Chapter 2. The resemblance becomes more evident when looking at the examples of documentation listed by the Linux Information Project (2006): quick start cards, manuals, books, computer-readable text (e.g. plain text files and web pages), audio and video, tooltips, intuitive design and comments included in source code. According to the Linux Information Project, the distinguishing factor between documentation and information is that documentation focuses on the technical aspects of an object or a system, while information may also cover e.g. the historical, business, economic, scientific, political, and philosophical aspects. However, an OSP **must** communicate clearly at least the business, economic, political, and philosophical aspects of its activities: these include information on the OSP's level of hybridity, community ideology, decision-making structures etc, which were discussed earlier in section 4.2. These are typically documented on the OSP's web portal. The problematics related to the use of the word *documentation* is further demonstrated by the labeling used on OSP web sites. For example, consider the information architecture of the "PostgreSQL Documentation" web page, as shown in Figure 13. below.

Figure 13.PostgreSQL Documentation web page at http://www.postgresql.org/docs/

The web page shown in Figure 13. has links named "Online Manuals", "frequently asked questions" (FAQs), "what's new", "books", "Technical documentation", and "Security Information". It is particularly interesting to compare what can be found behind the links "Online Manuals" and "Technical documentation": the "Online Manuals" comprise the "official" PostgreSQL documentation for both users and developers, while the "Technical documentation" link leads to a PostgreSQL Wiki page that is targeted to "PostgreSQL contributors" wanting to contribute "user documentation, how-tos, and tips 'n' tricks related to PostgreSQL". On the other hand, the PostgreSQL Wiki also includes a page that is targeted to developers and which contains information such as a developer FAQ, but to which there is no link from the "Documentation" page. Clearly, the "Documentation" and "Technical documentation" categorisation is artificial and meaningless. Furthermore, the PostgreSQL web portal demonstrates how difficult it can be to establish what counts as "documentation" and where goes the line between developer and user documentation. In my opinion, it would make more sense to simply talk about content targeted at different types of groups or audiences.

Matuska (2003, 35-36) divides open source documentation into two main categories: product-specific and project-specific documentation. The product-specific documentation describes the

software itself and its usage, while the project-specific documentation often describes the community and gives information on aspects such as project composition, workflow, internal guidelines, and standard procedures. Matuska divides product-specific and project-specific documentation into subcategories as shown in the table below:

Table 11. Subcategories of OSS documentation (Matuska 2003, 36)

Product-specific documentation	Project-specific documentation
User documentation Describes how an end-user can utilise the product. Examples of user documentation include installation and user's guides. The documentation may exist in various formats, such as web sites or traditional books.	**Documentation related to OSP structure and organisation** Documents the project-specific infrastructure and typology including, for example, roles and responsibilities; leadership and decision-making structures.
Developer documentation Describes the design, architecture, and functionality of the software to allow and advance its creation, modification, and development. The documentation may exist in various formats, such as comments in the source code or formal documents.	**Operational documentation** Includes operational guidelines for the project. E.g. defines the release process or the acceptance and integration process of new contributors.
	Project guidelines Standard operating procedures such as how to perform or document certain functions and activities. For example, how to inform other members about changes made to the source code.

Although Matuska's categorisation is useful for identifying the important dimensions in the information model, it also demonstrates how more emphasis is once again placed on code development, as project guidelines should also include guidelines for documentation developers and translators, among others. An open information model would also belong to this category as it describes the design, architecture, and functionality of the CMS and information architecture.

The observations made by Oram extend the scope of OSS documentation even further: Oram (2004) argues that in todays's world, contributing documentation can be as small an effort as asking a question on an online forum. He refers to this type of documentation as *community documentation*, *online (technical) documentation* (2004; 2006a; 2006b), or also as *free documentation* (2007). I have opted to use the term *community authoring* when referring to the phenomenon discussed by Oram for the following two reasons:

- to move away from the concepts of *documentation* and *information* and emphasise the role of *content* in the openness of OSPs
- to clearly relate *community authoring* with the *authoring and content-acquisition environment*, which is identified by Hackos (2002, 60) as one of the main components of a CMS.

Furthermore, I want to avoid the word documentation because of its strong connotation of printed text, which is inaccurate when talking about information that is distributed online.

As was pointed out by the Linux Documentation Project, Matuska, and Oram, OSS documentation and/or community authoring may present itself in many forms. In this study, the term *community authoring* covers various types of OSS-related information including commented code, web sites, mailing lists, WIKIs, issue tracking systems, IRCs, blogs, user-contributed tutorials, manuals, installation and user guides etc. I will next discuss the current role and characteristics of the new media typically used in OSS communities to communicate about OSS-related issues.

4.4.1 OSP portal and Wiki

For an OSP, its web portal represents what Stürmer (2005, 72) calls a "collaboration platform". According to Ye and Kishida (2003), an OSP "is unlikely to be successful unless there is an accompanied community that provides the [collaboration] platform for developers and users to collaborate with each other". The argumentation presented by Morville and Rosenfeld (2007,

460) further demonstrates the importance of the OSP web portal and its role both as a virtual meeting place and as an organisational infrastructure:

> Cultures and communities don't just happen; they require careful nurturing. On the other hand, they wither when overmanaged. A well-designed information architecture can help balance these two extremes, flexibly encouraging freedom of expression and action while organi[s]ing and structuring content for better findability. And where other information architectures have to fit within a context, an online community architecture *creates* that context — it is often the only place where its members meet.

The information architecture of an OSP is perhaps the most tangible snapshot of its organisational context including the mission, goals, strategy, staff, processes and procedures, physical and technology infrastructure, and culture of the project, as the web portal functions as a central point of access to all the relevant information and resources. The importance of the observation made by Morville and Rosenfeld is also echoed in the comments of Fogel and Hackos. Fogel (2005, 11) mentions that running an OSP "is partly about supplying information, but it's also about supplying comfort." A community-based information model can assist in supplying such comfort since it "not only facilitate[s] fast [information] search and retrieval, but also create[s] a sense of familiarity and belonging" (Hackos 2002, 131).

Hackos (2002, 124) has listed the tell-tale characteristics of a poor web site that lacks a logical framework, that is, an information model, for organising its contents:

- You can't tell how to get [...] to the information you're looking for.
- You click on a promising link and are unpleasantly surprised at what turns up.
- You keep drilling down into the information layer after layer until you reali[s]e you're getting farther away from your goal[...].
- Every time you try to start over from the home page, you end up in the same wrong place.
- You scroll through a long alphabetic list of all the articles ever written on a particular subject with only the title to guide you.

Such shortcomings are especially damaging for the web site of an OSP, because the web site has such a huge role in community building, as was discussed above. In addition to the more public and/or formal OSP web site, many OSPs also use a Wiki, which typically functions as a kind of a project intranet. Especially Wikis often suffer from navigational problems, inconsistent definition

of the target audience(s), and duplication of information (Fogel 2005, 52). The duplicate information may be a result of the first two problems, in other words, authors may rewrite information because they are unable to find the existing piece. A Wiki is a piece of software that allows anyone or only registered users to collaboratively create, modify, organise, and link the content of a website. Wikis use a very simple syntax that aims to make editing a web page or creating a new one as simple as possible. (Stürmer 2005, 21) Wikipedia[1] is probably the best known example of a Wiki. Today, there are a number of different implementations of Wikis available, but typically the software includes features such as a search function, change history, and revision control. Some OSPs use a Wiki to create draft documentation, while others may even use it as their primary documentation.

4.4.2 Mailing lists, discussion forums, IRC, and instant messaging networks

Stürmer (2005, 21) describes mailing lists as "a collection of email addresses of people who are interested in the same topic of discussion". The mailing lists may be open or closed, meaning that access to the mailing list may be granted by an administrator, or that people are free to subscribe to a list by themselves. The tools used to set up the mailing lists often vary, and thus so do their functionalities, but typically functions such as archiving and search facilities are provided.

Nevertheless, mailing lists are a very central communication channel for OSPs. Stürmer (2005, 94) who has studied OSS community building observed that mailing lists were the primary communication channel for all of the eight OSPs that he investigated. Berglund and Priestley (2001, 139) describe the benefits of mailing lists as follows:

> Discussion forums and mailing lists provide a high degree of user control, flexibility, and openness to contributions. The members of the community [can] easily participate. User control over content is built in to the submission structure. Release cycles can be very short as answers to questions are posted often within hours. Splits are not uncommon into different strands of continued discussion.

1. http://www.wikipedia.org/

Internet Relay Chat, or IRC in short, is an internet technology that allows both private talks one-to-one and group chats (or conferences) in so-called chatrooms. The chats occur in real-time, in synchronous manner, but can also be logged and published e.g. on the OSP website to increase availability. (Stürmer 2005, 21) Some OSPs may have multiple IRC channels relating to subtopics such as installation or development issues. Thus, a person can choose those channels that s/he finds useful or interesting. (Fogel 2005, 50)

Rothfuss (2002, 143) stresses what an important role IRC often plays in enabling social exchange between community members. IRC is used, for example, to conduct meetings, discuss development issues with other developers, and answer support questions asked by end-users. Stürmer's (2005, 92-93) study further demonstrates the importance of IRC as an OSP communication channel. One of Stürmer's interviewees referred to it as the "backbone" of his OSP. Another interviewee compares the role of mailing lists and IRC as follows: "[...] IRC is for very quick and simple questions and answers, the mailing list is to discuss issues in detail with the entire community [...]".

Clearly, mailing lists and instant messaging networks such as IRC compensate for the inability of the members residing in various locations to converse face-to-face. Mailing lists and online discussion forums are targeted at various types of audiences. The number of included mailing lists and IRC channels differs between OSPs, and so does the way they are targeted towards the different community members and how they account for the different roles. These forums serve many purposes such as project management, technical support, product management, and even marketing and public relations as notifications about events and new releases are typically also announced on mailings lists.

The main problem with IRC, mailing lists, and online discussion forums is that they are unscalable communication models (Fogel 2005, 91). Fogel estimates that the mailing lists of a project can support approximately "up to a few thousand users and/or a couple of hundred posts a day". Consequently, as Fogel (2005, 16) notes, "[a]t the early stages of a project, there's no need

to have separate user and developer forums". However, as the amount of traffic increases, new, more specialised forums typically need to be set up. Moreover, the same logic applies to IRC channels and online discussion forums. (Fogel 2005, 91)

Berglund and Priestley (2001, 139) describe the shortcomings of these communications media as follows:

> Unfortunately, discussion forums and mailing [...] lists have difficulty supporting the task of building documentation. Extraction of material into documentation is seldom performed, making discussions concerning topics difficult to track. Search engines do not exist for such purposes, but require a common terminology across submissions and support only active search (not passive browsing).

Furthermore, Oram (2006a) argues that mailing lists can also hinder the learning process of OSP members. Oram calls the problem "give a man a fish" behaviour: when a person asks for a solution to a problem, s/he may receive a piece of advice that allows him/her to fix the issue, but does not explain the phenomenon that initially led to the problem. Or, in terms of the continuum of understanding, the received information does not contain enough context that would allow the person to transform the information into knowledge, and ultimately to wisdom which might enable him/her to understand and fix future problems by him/herself. To a certain degree, the same may hold true for IRC as well.

As the amount of traffic on a forum increases and need to create new forums becomes more evident, the OSP administrator needs to decide which dimension to use to isolate forums. The forums are typically set up according to actors and/or roles: there may be a user forum, a software developer forum, and translation/documentation forums. This categorisation may not, however, be optimal: it tends to isolate the programmers, authors, and translators. Thus, also the dimensions for online forums deserve to be tackled in the information model to ensure that important information reaches all relevant actors.

4.4.3 Software configuration management (SCM) tools

Software configuration management encompasses a set of tools (revision control systems, also known as Version Control System (VCS), or (Source) Code Management (SCM)) used to store and maintain software source code and the associated files and track their evolution (Jones 2006; Rothfuss 2002, 130). As was noted earlier in section *2.2*, software configuration management is another facet of information management just as content management. SCM is used to store files and their revisions in a repository, detect conflicts in changes made to the source code, notify developers of the conflicts, or automatically merge unconflicting changes made by different developers to the same piece of code. SCM also allows the files that make up the source code to be grouped together into sets to create consistent and repeatable software builds (Jones 2006).

Revision control means that all changes made to the source code and also the originators of the changes (i.e. the person who *committed* the code change to the repository) are recorded (Jones 2006; Stürmer 2005, 22). Revision control systems are a central prerequisite to the openness of the OSS development process: they provide read-access to everyone interested in reviewing the code and enable several developers to work on the same software components simultaneously (Stürmer 2005, 22). Consequently, revision control is also an enabler of parallel, open development as defined by Ye and Kishida (2003), that is, providing access to interim development versions of the source code (see *4.1*). On the other hand, write-access to the source code is typically only granted to developers who have already proved their talent. Newcomers often only write patches, i.e. small code changes saved in files, which the developers with write-access commit to the repository. Moreover, the revision control system allows developers to cancel a bad code commit and revert back to a version of the source code that was known to be stable. (Stürmer 2005, 22)

SCM tools are often integrated with issue tracking systems and mailing lists. For example, every commit made to the source code repository generates an email that is sent to a dedicated to code commits. The email notification shows who committed the change, when the change was made, what files and directories were changed and how. As the other developers have also subscribed to

the commit email mailing list, the commit notifications enable quick and systematic code reviewing. The code commits mailing list is also one of the most important ways to inform developers about what is going on in the project at code level. (Fogel 2005, 42)

4.4.4 Issue tracking systems

OSPs typically use an issue tracking system to monitor and administer issues such as feature requests, software defects (bugs), tasks, patches etc (Fogel 2005, 46). As Fogel mentions, tracking systems (also called bug trackers, ticket systems, etc) can be used to track "anything that has a distinct beginning and end states, with optional transition states in between".

Today, there are many issue tracking systems with various different features to choose from. The issue trackers may, for example, support functionality such as integration with web and email interfaces and software configuration tools (Rothfuss 2002, 144). Consequently, issue tracking systems are a central tool for many OSS development activities ranging from debugging (identifying and resolving software defects), to requesting new features, project management, and assigning development tasks to contributors. In fact, in many cases issue tracking systems can also be considered to incorporate a content management system that is used to collect, store, and distribute information about project-related activities. JIRA[1] is one example of an issue tracking system used in OSPs. It is widely customisable, and includes features such as:

- managing all kinds of project-related issues including bugs, features, tasks, or enhancements
- defining project components and versions
- managing workflows i.e. the set of steps and transitions an issue goes through during its lifecycle
- managing users, groups, and project roles: for example, assigning members to project roles
- sending email notifications to users whenever an event that is relevant to them occurs
- compiling release notes for a project from a set of related issues

1. http://www.atlassian.com/software/jira/

- integration with other tools and systems such as revision control systems.

The issue tracking system of an OSP is not only useful for the contributors but also for observers of the project, that is, individuals and/or organisations evaluating the usefulness of the software. As Fogel (2005, 16) notes, "an accessible bug database is one of the strongest signs that a project should be taken seriously [...] the higher the number of bugs in the database, the better the project looks". This may seem highly controversial, but the logic behind this observation is that the number of identified issues (which also include feature requests or pending tasks, not only defects) depends more on the amount of users and how easy it is to file an issue than it does on the absolute numbers of defects present in the code. Issue trackers represent "a public face of the project as the mailing lists or web pages" (Fogel 2005, 48).

Despite the wide array of features and the possibility for fine-tuning, according to my personal experience issue trackers are typically not personalised based on an information model. Issue tracking systems typically suffer from duplicate issues, as users have trouble sorting through a large amount of issues. Issue trackers are often regarded primarily a tool for software developers, but, as noted by Fogel, issue tracking systems are also used by OSS evaluators and users to assess the activity and "liveliness" of the project. Issue tracking systems are normally based on forms-based authoring, which is highly structured, and is thus suited to be implemented in XML. Consequently, they could be used effectively to create content of high granularity and the content units could be reassembled to create many different types of information, such as release notes.

4.4.5 Blogs and planets

Stürmer (2005, 21) defines a blog as a "web application that manages the frequent posting of new messages". Furthermore, people typically use blogs as an online personal journal to publish their own thoughts, opinions, comments, and links: the introduction of this study showed one real-life example of a blog entry (Figure 4.). Popular OSPs often have their own, OSP-specific blog, or an OSP may host a so-called *planet* which collect posts from the blogs of OSP members

and displays them on a single web pag (Stürmer 2005, 21;95). Blogs normally also support interactive communication between the author and the readers as they allow readers to post their comments after a blog entry (Stürmer 2005, 21-22).

Another one of Stürmer's (2005, 96) interviewees makes useful observations of the essential differences between mailing lists, blogs, and planets. Planets allow you to "feel the pulse of the community", that is, they give you a sense of the general atmosphere, or the attitudes of the members within the community. They give an "abstract overview" about what is going on at the moment. The interviewee adds that blogs "usually have a broader spectrum of readers than the project mailing list". Thus, a planet normally reaches a wider audience.

4.5 Information needs according to OSP stage

In this section I will first relate the information needs to the different stages of the OSP life cycle. As was noted earlier in section *4.2.1*, the difference between the alpha and beta stages is often just a matter of judgement. I have therefore loosely divided the information needs according to:

1. information needs in a pre-alpha, alpha, and beta stage OSP
2. information needs in a stable or mature OSP.

Next, I will build a vision of the experience that the actors representing the OSS community have with the current information. According to Hackos (2002, 37), this means "gathering information on the current user experience by understanding the successes and failures of that experience".

4.5.1 Information needs in a pre-alpha, alpha, or beta stage OSP

In a pre-alpha or alpha stage OSP, some preliminary source code has typically been released, thus the project is considered to have been launched. At this stage, the development activities focus at fixing the most serious defects and adding new features. As a result, the project gradually begins to take shape and the software starts to be functional at least some of the time. Raymond (2001) summarises the main requirements of launching a new OSP as follows:

> When you start community-building, what you need to be able to present is a 'plausible promise'. Your program doesn't have to work particularly well. It can be crude, buggy, incomplete, and poorly documented. What it must not fail to do is convince potential co-developers that it can be evolved into something really neat in the foreseeable future.

Many researchers argue that it is the source code that plays the most central role in creating such as first impression, but, interestingly, Fogel (2005, 27) tells a different story. Subversion, a project Fogel has been involved in for about five years, started with "a design document, a core of interested and well-connected developers, a lot of fanfare, and no running code at all". He adds that the extremely popular Mozilla project also began with no running code. According to Fogel (2005, 11) the most difficult task in launching a new OSP is "transforming a private vision into a public one." Sometimes words are better able to visualise the potential of a new solution than buggy, incomplete code. Consequently, one might argue that for a planning or pre-alpha stage OSP, content may be even more important than code.

This applies especially to the content of the project web site where appearances certainly matter. It is often the project web site that people first stumble across and on which they base their conception of what the OSP is about. Readers are quick to assess the appearance and content of the site and to determine just how much thought and effort went into planning its presentation and structure. At the initial stages, the web site needs to include enough useful information for newcomers so that they will be able to easily pass the initial hurdle of unfamiliarity with the project. (Fogel 2005, 10-12) According to Fogel (2005, 16), "[a]mong early adopters, the distinction between developer and user is often fuzzy". Furthermore, the ratio of developers to users is typically higher at the early stages of the project.

Fogel (2005, 13-18) also gives quite concrete advice about the kind of information that should be provided right from the very start. The first thing that people see should be a mission statement of the project, which allows them to decide within 30 seconds if they want to know more or not. The mission statement must be concrete, concise, limiting, and, above all, short. The second most important thing is to remember to state that the project is open source and the licence that is used.

The web site should also describe the supported, planned, or pending features, and the required computing environments. The development status is also important: while this may be difficult to pinpoint at the initial stages of the project, it is nevertheless useful to explain the short-term goals and development needs as they provide a simple point of entry for potential developers who might be interested in contributing.

As for the software itself, it must be downloadable as source code in standard formats. The latest source code needs to be available in real time, with unambiguous information about the changes made and the latest version. Fogel (2005, 15-18) According to Fogel, binary (executable) packages are not mandatory at the initial stages of the project, unless the software has very complicated build requirements or dependencies that could thus present a substantial obstacle for new users. On the other hand, it is extremely important that some instructions are included about how to install and set up the software and how to test it to confirm that the set up was successful. It would also be good to have at least one tutorial of how to perform a common task.

To allow a smooth transition from project evaluators and/or users to developers, some basic developer guidelines should also be provided. Setting up of communication channels such as a mailing list is naturally an absolute requirement. (Fogel 2005, 16-17) Fogel (2005, 29) suggests the following as a minimal set of required tools for managing information:

- web site
- mailing list(s) and a real-time chat
- version control
- issue tracking.

Canned hosting (i.e. using a site that provides free hosting and infrastructure for an OSP including a web area, revision control, an issue tracking system, a download area, chat and mailing lists, regular backups etc.) can be sufficient at the early stages of the project (Fogel 2005, 20; 54).

Although an extensive content management system might be an overkill solution at the early stages of a project, it would be beneficial to allocate some resources for building a small-scale

information model to define some basic information needs, information types, and authoring guidelines, and thus to prepare for possible future expansion in project size which might demand the adoption of a content management system. One of Stürmer's (2005, 50) interviewees pointed out another important aspect worth documenting at the early stages of the project: "[o]ne other thing that should be thought of in the beginning is the basic API".

As Rothfuss (2002, 54) observes, "[m]ost OSP[s] start with ad hoc coordination". However, as a project grows, the need for infrastructure and project management increases. While a creative chaos may provide a rich soil for creating and exchanging ideas, it can soon turn into an impediment as the project enters stable or mature stage. The lack of coordination soon becomes a barrier of entry for new participants unfamiliar with the informal ways of coordination and information exchange. The information needs and content management requirements in an stable or mature OSPs are discussed in more detail in the next section.

4.5.2 Information needs in a stable or mature stage OSP

As was discussed in Chapter 4, when the software reaches stable stage and has become reliable enough for production/daily use, it is able to gain more normal end-users, who are also potential future contributors. Consequently, as an OSP gains popularity, the number of information seekers increases dramatically in the online forums, but the number of knowledgeable people able to respond to the incoming flow of questions grows much more slowly. (Fogel 2005, 91) End-users start to outnumber developers. The OSP may soon be faced with a scalability problem related to handling communications.

This scalability problem can turn into a barrier of entry for newbies who cannot find the answers to their questions. This in turn can increase the frustration of developers who have been involved for a long time, but who may not be consciously aware of the accumulated body of tradition and how difficult it can be for a newbie to acquire all the related information. It may seem to the developers that new community members are "dumbing down"; that they keep asking

the same questions over and over but still learn nothing. (Fogel 2005, 94) The following real-life experience described by Rothfuss (2002, 93-94) describes the problem perfectly:

> For the time period from July 2001 to August 2002, I contributed to the PostNuke project. I spent about 40 hours a week helping to develop the core architecture of this community-oriented Content Management System. A very enjoyable experience and I learned a lot in a short amount of time about software engineering, interacting with large virtual groups and conducting a dialogue across language barriers and time zones. The project, meanwhile, exploded in popularity, and attracted thousands of users. To date, the software has been downloaded over 500'000 times, and been installed on tens of thousands of web sites. This popularity led to an interesting phenomenon. The more users the project attracted, the more newcomers that knew little or nothing about the conventions of the Open Source community assaulted the social fabric that had held the project together. The common courtesy among fellow developers made room for clamors for more, better, faster, and appreciation declined. Cries of conspiracies emerged, and many developers grew increasingly frustrated with the monster that they had created. Andy [...] summed the feelings of many longtime contributors up as "users are a pain in the ass". The situation got so bad that most contributors quit in the same week. What had led to this rapid burnout?

As Fogel (2005, 10) explains, the provided information "[...] should subscribe to the principle of scaled presentation; that is, the degree of detail presented at each stage should correspond directly to the amount of time and effort put in by the reader. More effort should always equal more reward. When the two do not correlate tightly, people may quickly lose faith and stop investing effort." Clearly, the developers of PostNuke felt they were rather being punished for their efforts. The OSP must be able to record some of the relevant information to take some of the unnecessary load off the mailing lists and IRC channels. Boiko (2005, 189) recognises the same dilemma: "If a community site is to draw a wide member base, it must provide enough relevant and accessible content to warrant attention. To hold the attention of the community, the provided content must continually grow and deepen." Both Boiko (2005, 189) and Hackos (2002, 33) argue that this can only be achieved using a CMS. As was described in the earlier sections, OSS release cycles are typically short and development happens in parallel. Moreover, in a stable project several versions of both the software and product-related content are maintained simultaneously. This leads to continuous publishing of new information, which:

> [...] requires not only that information be updated in a timely manner but that new information be continually made available. It means that information created by users is fed

> back into the system and made available to others in the user community. The only possible way to accommodate dymanically changing information is through a sophisticated content-management solution supported by an Information Model that anticipates continually changing information resources. (Hackos 2002, 33)

Thus, it can be concluded that an OSP should start investing resources in a CMS as it reaches stable stage, at the latest.

As the project matures, the role and importance of the OSPs web portal increases. A well-designed portal, the information architecture of which has been derived from an information model, becomes an absolute requirement. If the project was launched using a canned web site, it soon becomes inevitable that the project will need a hosting service of its own as the community scales up and more personalisation, customisation, and sophistication is required. As OSPs grow, they need to:

> [...] mobili[s]e other skills than programming. Documentation for developers, for example, is a crucial part of the effort to lower the barriers to participation for new volunteers. Translations of documentation and of program interfaces play a crucial part in program's dissemination. In addition, as F/OS reaches a wider, non-technical, user base, questions of usability and design are becoming more crucial. The issue of how relations between these different types of contributors are organized, therefore, will become increasingly important. (Berdou 2007, 19)

Furthermore, the contents of the web portal are the best marketing media for attracting users and contributors but can also help limit the amount of uninvited interest (such as the feedback received by the PostNuke developers):

> The vocabulary and structure of your web site and your intranet is a major component of the evolving conversation between your business and your customers and employees. It influences how they think about your products and services. It tells them what to expect from you in the future. It invites or limits interaction between customers and employees. (Morville and Rosenfeld 2007, 26)

It is important to state, for example, the expected technical skill level of the audience (Stürmer 2005, 51). Mature projects should also inform the audience about "how actively the project is maintained, how often it puts out new releases, how responsive it is likely to be to bug reports, etc" (Fogel 2005, 14). This information could be published, for example, on a development status page that contains a history of past releases with feature lists.

4.5.3 Actor-specific information needs and user experience

In this section, I will aim to build a vision of the actors' experience with OSS-related information, both within and without an OSS community. I have decided to focus mainly on actors and not roles firstly because my goal is to build a general-level information model. I have chosen to look at the information needs of project evaluators, newbie contributors, and developers.

In section *4.2.3* learning was recognised as one of the main motives for participating in open source development. Furthermore, learning practises and processes, for example peer-review, also represent "constitutive elements of the open source development model" (Berdou 2007, 111). However, as Berdou (2007, 113) observes, despite this open and collaborative character, OSS communities are often characterised by significant barriers to entry.

Berdou (2007, 59; 61) interviewed 23 contributors from the KDE and GNOME projects, which she describes as "mature projects in both a technical and a communal sense and have developed significant ties with the commercial world". Among the interviewees were both paid and volunteer programmers, non-programmers, and newbies. Several of Berdou's (2007, 113-115) interviewees associated the barriers of entry with inaccessibility of the OSP's information resources, which hinders the learning process of newbies and may thus prevent them from becoming contributors. According to Berdou, "the most common problem in relation to learning concerns the fragmented character of the documentation and other online resources that were available". Especially information about conceptual aspects related to understanding the development process or program architecture may often not be provided at all. In fact, Berdou observed that a process she calls "lurking" on the mailing lists and IRC channels was the best way for a newbie to find a suitable development task. Berdou's interviewees described their integration to a open source project as a "slow learning process, during which they build up skill sets and community knowledge and position themselves in developments by choosing suitable tasks". This shows that issue trackers are often not used to their full potential to indicate open tasks to volunteers.

Berdou (2007, 115; 127) found that experienced developers, on the other hand, tend to stress the importance of self-reliance and expect newbies to orient and teach themselves using whatever learning resources are available. New contributors are expected to gain an understanding of the dynamics and practises of the project before seeking the help of experienced developers. Experienced developers sometimes see the barriers of entry as a necessary step in the initiation process of new members that is meant to banish the ones who are not serious in becoming involved. "Every minute helping a newbie is a minute spent away from writing code", and thus holding the hands of a newbie not wanting to commit to the project is a bad investment (Berdou 2005, 115). Consequently, it can be concluded that the experiences of new members and experienced developers concerning the published content may sometimes conflict.

Such barriers of entry are lower in a new or immature project, than in a more established one. Firstly, there are fewer lines of code to read, and thus the existing software architecture may be more discernible from the code. As Mork (2006, 56) observes, "[o]pen source code is based on the principle that code can speak for itself. While intelligently written code has some ability to speak for itself, being a masterpiece doesn't necessarily make it easier to understand. Documentation inside the code is therefore easy to use if you know where to look, but is in itself insufficient." Secondly, there are fewer established traditions and practises in an immature project, and new members are better able to suggest new and better ones. Thirdly, as noted by one of Stürmer's (2005, 16) interviewees "it's much easier to get into a project where not much has happened so far. There are still things to do on a relatively low level". Furthermore, Mork (2006, 12) argues that certain aspects of software architecture are important for adopters and developers alike. These include "design patterns, rationales for design choices, performance and scalability of modules, and API compatibility and adherence to standards." Understanding the software architecture is, for example, "the key to understand performance and reliability."

4.6 Summary of current shortcomings and recommendations based on needs assessment

To conclude the first phase of the content management design process, needs assessment, two deliverables should be produced. These include a report and a recommendation, which:

- define the business problem at hand and how its solution will impact profitability
- specify the business case to "show what it costs to continue handling content as it is done today, what the short-comings are of the current solution and what efficiencies and cost savings might be reali[s]ed with a new and better solution". (Hackos 2002, 38)

Barriers to adoption of open source, barriers of entry, and "openness to join"

The discussion has shown that providing relevant, accurate, and accessible content is crucial for the success of an OSP. The role of content management is especially significant for a project that is rapidly increasing in size and complexity. Lack of high-quality content can in the worst case scenario turn into a vicious circle as the project may be perceived as immature, inactive and thus is likely to attract new contributors. Furthermore, insufficient content management and lack of an information model is not only a question of image, but it can also make the project unapproachable to new contributors, while long-term developers may not be able to recognise the problem.

In the near future, there will be more and more demands for efficient information development solutions and content management systems within the field of OSS. As the number of OSPs increseas, the competition for new contributors is becoming more and more fierce. Furthermore, new projects are no longer started by "lone programmers" but increasingly often by institutions that have more resources to put into a project startup (Fogel 2005, 10). A lone programmer wanting to create a successful project will need to have even better networking and marketing skills to be able to promote his/her project in similar scale. Lastly, we are likely to see an invasion of new users/contributors coming from Japan and the third world, especially China and

other Asian countries where the role of English is less important. These trends make content management, translation, localisation, and internationalisation even more important issues for the success of an OSP.

Benefits of open information models

One important benefit of open information models, metadata standardisation, was already touched upon in Chapter 2. Metadata standards can bring significant efficiencies in content creation, sharing, and reuse, but can also also benefit content usability. In other words, metadata standards can make information reuse more efficient not only within an organisation but also across an industry. The field of open source holds even more potential for such standardisation and improvements in efficiency because of its endeavour for openness.

As Hackos (2002, 192-193) points out, little progress has been made toward standardi[s]ation in the high-tech industries". As a result, organisations use numerous different ways to organise and structure their information. When considering the effects of presenting information in a variety of different ways from an end-user's point of view, it can be noted that related products and thus also related information are often used side by side by the same individuals. The end-user is not, however, served by this confusion. According to Hackos (2002, 35),

> Standard terminology for identifying information will allow authors to label information more consistently and make it easier to retrieve and deliver. [...] Especially important is the need to negotiate the terminology used to tag information. If the terms are ambiguous or poorly defined, searching for tagged information may prove frustrating for all levels of user, both internal and external. Although terminology decisions require careful analysis and frequent compromise, they are critical to the success of information integration.

The end-user would be better served if, for example, all OSPs within the same industry (e.g. all open source content management systems) used similar categorisations and terminology on their web sites. Furthermore, as Hackos (2002, 193) observes, web delivery tends to highlight the lack of standard metadata, terminology, and taxonomy. For example, one OSP may provide information about bug reporting guidelines behind a link named "Documentation" while another OSP might have placed this information behind a link named "Contribute".

There is, in fact, already an open source, XML-based standard available for topical writing and information typing called the Darwin Information Typing Architecture or DITA in short. In DITA, topic is the smallest maintainable unit of reuse. DITA allows authors to write three types of topics: concepts, procedures, and references. DITA authors can also create their own, specialised information types, if need be. (Hackos 2002, 194-195) Surprisingly, however, DITA seems to be somewhat unfamiliar among OSP developers: for example, a Google search using the words "DITA" and "open source documentation" produces only 137 hits, and many of the results point to the article by Bergman and Priestley published already in 2001. During the writing of this study I also had a look at some established OSPs including, for example, Plone[1], MySQL[2], PostgreSQL[3], Apache Derby[4], OpenOffice[5], and Cocoon[6]. Of these four projects, Apache Derby was the only one to use DITA.

Standardisation and use of XML can bring efficiencies through both intra- and inter-organisation collaboration on content management. However, as Hackos (2002, 197) points out, "information types alone will not provide enough standardi[s]ation for highly technical information delivery. The information developers must go inside the information types to standardi[s]e the content units that are the building blocks of the information types."

But the use of open XML standards such as DITA and open information models can also bring other benefits for OSPs apart from efficiencies achieved through collaboration. As has been mentioned earlier, the ability of contributors to freely choose from a variety of tasks of manageable size is one enabler of "openness" in OSPs. In software developement, this is achieved through modular software architecture. Similarly, by using DITA XML in content authoring, writing tasks can be broken down to the level of topics: the use of standard topics, information types, and content units ensures that two topics written by two different authors have the same

1. http://plone.org/
2. http://www.mysql.com
3. http://www.postgresql.org
4. http://db.apache.org/derby/
5. http://www.openoffice.org
6. http://cocoon.apache.org/

content, organisation, and structure. But use of XML also helps separate form and content. When unstructured authoring is used, the author needs to worry about the formatting as well as the actual contents. Furthermore, the task of building a web site for an organisation is even more complex and multifaceted, as the following figure demonstrates.

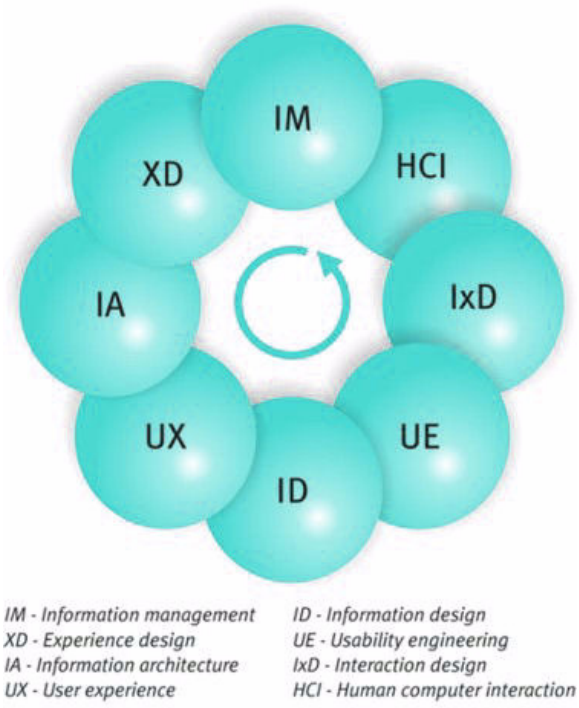

Figure 14.The eight flavours of information architecture (Kennedy 2007)

Use of XML also better allows to separate between the different areas shown in Figure 14. and the roles of an information architect, usability expert, information designer, application-programming specialist, and content editor. This in turn would support the goal of lowering the barrier of entry and providing a wide range of relatively small tasks that volunteers with different backgrounds are free to choose from.

5 Defining an information model for open source

The following list outlines the steps of creating an information model as defined by Hackos (2002, 39-43)

1. Define the dimensions of use for identifying the main categories of the information model. These categories represent the outermost layer in the three-dimensional information model[1].
2. Identify the information types to be used. The information types represent the middle layer of the information model.
3. Identify the content units that authors will use to construct the information types. The content units represent the third, innermost layer of the information model.

As Hackos (2002, 145) emphasises,

> One information model for an entire corporation is a significant undertaking, requiring multiple levels of information gathering. It requires coordination among many individuals and agreement on the dimensions needed to describe author, information, and user requirements.

Thus, I recognise the impossibility of being able to cover all the details of designing an open source information model in one study. Furthermore, one might even question the whole approach of trying to define an information model for an entire field, particularly one as varied and diverse by nature as OSS. After all, Hackos underlines that it may often prove necessary to create several information models just to tackle the information flows of a single organisation.

Nevertheless, I believe that there are certain important OSS-specific dimensions, information types, and content units that can and **should** be identified in a general-level information model. Furthermore, during the writing of this study, it became evident to me that there are huge gaps within contemporary research about OSS, related to content management, technical communication, and community authoring. As I progressed with my research, the aim of my study expanded into an attempt to use Hackos' framework to identify future themes for research as

1.The three-tiered information model is depicted in Figure 10.

much as document existing trends and good practices, as was already mentioned in the Introduction. As I fully realise the complexity of the task at hand, I will, apart from a few exceptions, maintain the analysis at the level of dimensions of use and information types, not content units or content plans. I will first discuss the results of the information needs and resources analysis (Chapter 4) to identify a set of main dimensions for OSS content management and use them to divide the information needs and/or resources into several smaller and more manageable "content areas", as I decided to call them.

5.1 Dimensions of use

The dimensions of use represent metadata that will be used to label the content and make it modular. The dimensions can be based on:

- business information requirements, such as different product types and models, market segments or subject matter
- user requirements, such as user job, skill level, experience, language, country, etc.
- author requirements, such as author, title, ID, editor, approver, original date, revision dates, version number, etc.
- relevant use cases or scenarios of use. (Hackos 2002, 39-40)

Table 12. shows the main dimensions that I will use to distinguish between content areas.

Table 12. Main dimensions for open source information model

Actor	Product-related content	Project/community-related content
Evaluator (corporate, institutional, private)	"Marketing material"; overview of basic functions/features, white papers, licencing, support etc	Size, status, maturity, activity, reputation, related projects, etc
Inactive user	User guides and manuals, contextual help, support forums, user interface and error messages, etc	-

Table 12. Main dimensions for open source information model

Actor	Product-related content	Project/community-related content
Reader, contributor	Product design and architecture specifications, technology specifications, interface specifications (API, other OSS) etc.	Typology, infrastructure, processes, tools, forums, developer guidelines, project and issue management, related OSPs

Table 12. is based on Matuska's categorisation (presented in Chapter 4), but I have modified it by adding the "Actor" column to the categorisation. My categorisation of "OSS actors" is loosely based on the discussion included in Chapter 4. The "actor" dimension helps include non-coders and their information needs to Matuska's categorisation, but it also clarifies the differences between each content category.

Based on Table 12., it is possible to distinguish between five different main areas of content:

- product-related content for evaluators
- product-related content for inactive users
- product-related content for readers and contributors
- project/community-related content for evaluators
- project/community-related content for readers and contributors.

End-user documentation, that is, product-related information targeted at inactive users will, however, not be discussed further as it is considered to be quite similar to user documentation of closed source software.

The following figure shows the main page of the OpenOffice project, which uses a categorisation similar to the one presented above.

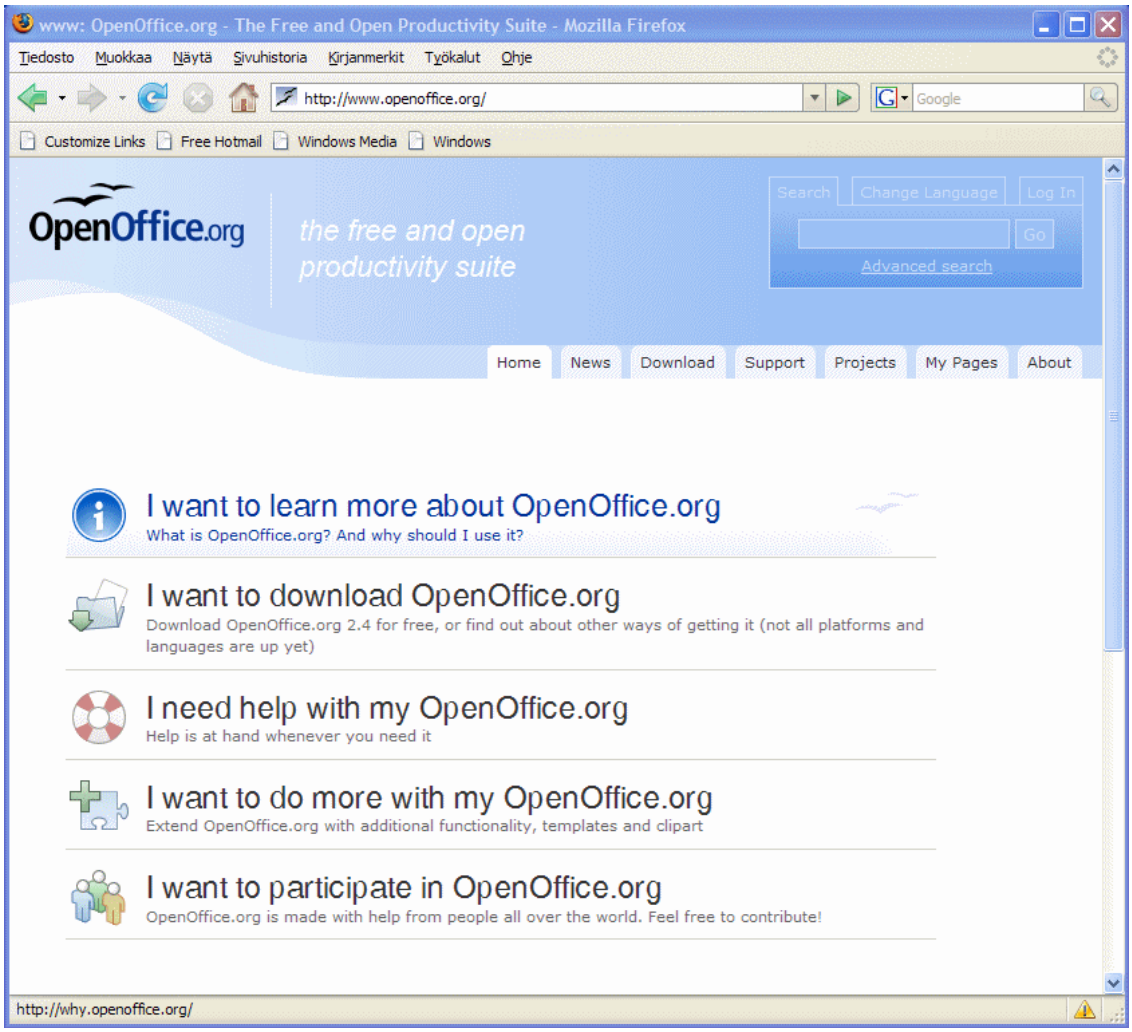

Figure 15.OpenOffice.org main page at http://www.openoffice.org/

The first link "I want to learn more about OpenOffice.org" is for evaluators, the next three links are for end-users and the last link is for project volunteers.

5.1.1 Product/project-related content for evaluators

The first content areas to be discussed are project and product-related content targeted at evaluators. In addition to evaluating the open source product itself, evaluators often also want to assess the maturity and activity of the project. The decision to download the product may depend on the size of the community, which in turn may have an impact on the ability of the community to support fellow users when a user hits a bug, for instance. Most of the project-based content for evaluators is relatively static by nature, i.e. the information would not need to be updated in real-

time. It is also very project-specific, in other words, it is unlikely that another OSP would want to reuse this content as-is. Nevertheless, while the contents of the files themselves may not be reusable, the categorisations, that is, the dimensions or metadata defined in the information model are reusable because OSPs within the same field i.e. producing similar products are likely to have evaluators with similar profiles and information needs.

Evaluators and/or end users are commonly categorised into subcategories, such as:

- beginners, advanced users, and expert users
- corporate, institutional, or private users.

The first categorisation is typically useful when dealing with end-user documentation such as user's guides and will therefore not be discussed further here. The second categorisation, however, can be relevant when authoring content for OSP evaluators as an evaluator may not always be an individual but also an institution or organisation, who might e.g. be thinking about investing resources to the project. Consequently, it may be useful to author personalised content to attract investors, for example.

The second categorisation can also be used with product-related content targeted to evaluators: just as commercial companies, also OSPs need to provide information about licencing and basic product functionality. While commercial companies typically publish such information in marketing brochures and white papers, OSPs often deliver this information on their web portal. Figure 16. below demonstrates how the OpenOffice site takes advantage of this categorisation.

Figure 16.Personalised content for open source evaluators at http://why.openoffice.org/

Each icon (Governments, Education, Businesses, etc) in Figure 16. represents a link to personalised product-related content, taking into account the different backgrounds of the different evaluator groups.

5.1.2 Product-related content for readers and contributors

This content area includes content targeted at software developers, consisting of software descriptions and API reference. The requirement for openness of this content is lesser in company-based open source than in volunteer-based open source. Because of the wide range of open source products, it is difficult to identify detailed dimensions for this content area. Typical choices might include subject areas such as software components or programming languages.

5.1.3 Project/community-related content for readers and contributors

The first content area was concerned with an outside view of the OSP, where the project and/or the related community is seen as a closed unity. The next content area is much more complex, as a great number of interrelated dimensions can be identified. The dimensions are related to software

versions, user requirements (actors, roles, skill level, language, etc), author requirements (e.g. skill level), and scenarios of use. The following figure aims to demonstrate some of these dimensions.

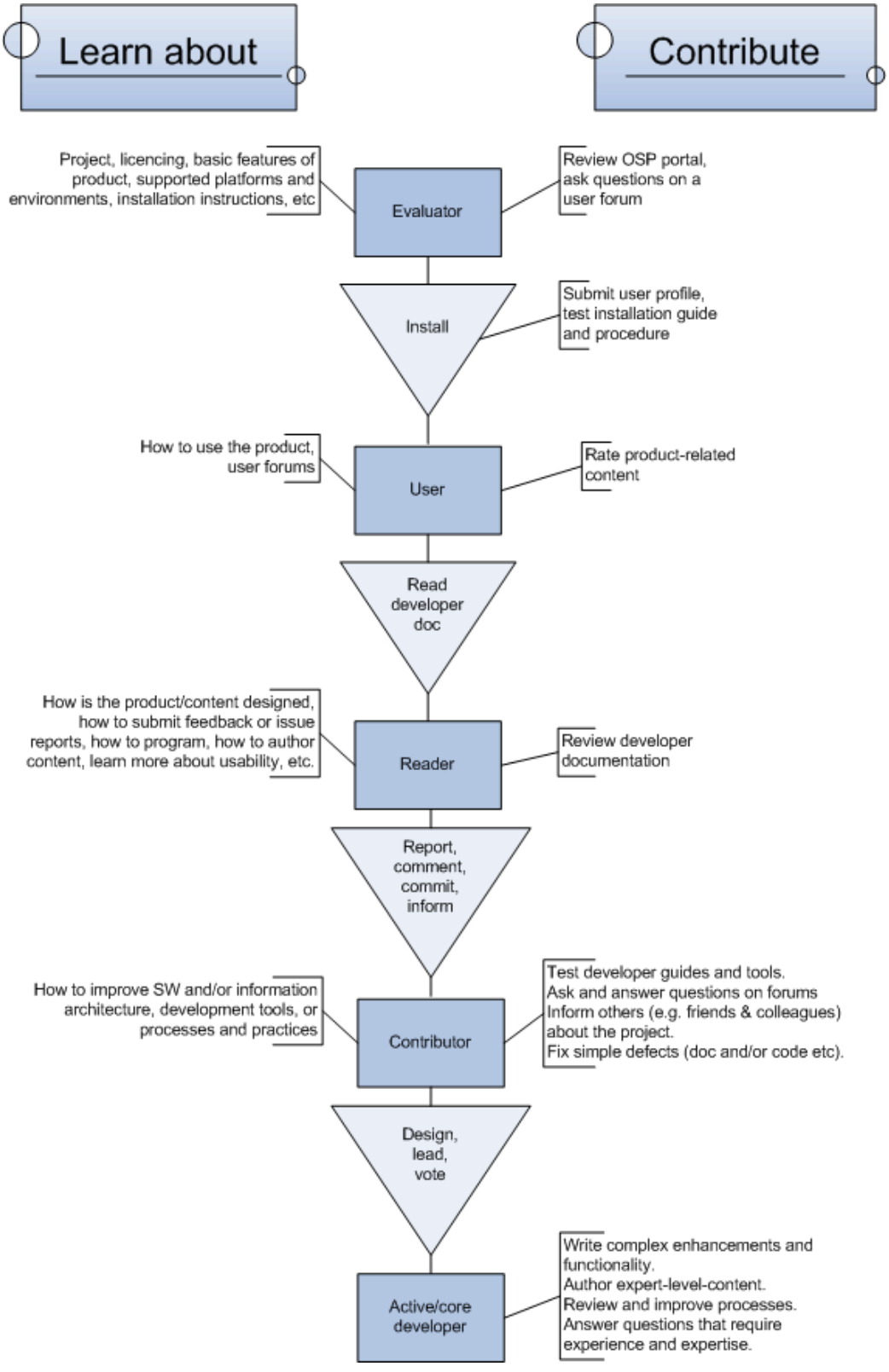

Figure 17. Basic dimensions of OSS development

Firstly, Figure 17. shows the two main scenarios of use to be discussed in this study: learning and contributing. Learning and information/knowledge sharing was identified as the main motivation for participating in an OSP, while community building i.e. acquiring new users and contributors was recognised as one of the major goals of an OSP. Thus, the two scenarios of use, learning and contributing, are closely interrelated. The information model should also support the goals of learning and community building and aim to remove or at least lower any possible barriers of entry met by potential volunteers.

The second important dimension, actors, is also presented in Figure 17. I have based the categorisation of actors on Ye and Kishida's (2003) onion model (presented in Chapter 4), but have omitted some actor profiles for simplicity. Figure 17. aims to show both the information needs from the point of view of an individual and also the potential information resources from the community point of view. Furthermore, it shows how an individual "climbs up the ladder" as s/he decides to become more involved with the project. I also wanted to demonstrate that even evaluators, inactive users, and readers contribute to the project because they are in effect testing the usability and accuracy of the OSP portal as well as the installation procedure and user/contributor instructions — the central question is how their contributions and experiences might be captured, made visible, and transferred to the OSP's knowledge base. Consequently, as was discussed in Chapter 4, there is a very thin line between the actor profiles of inactive user, reader, and contributor. "Readers" are an important group to be taken into account when defining this content area in the information model. Just as evaluators are evaluating the benefits of downloading the product, readers are evaluating the costs and benefits of becoming a contributor. The benefits may range from learning and personal growth to monetary rewards, while the costs can typically be translated as the time and effort that needs to be put into a contribution. A well-designed information model can lower the costs by improving the relevancy, accuracy, and accessibility of information. Furthermore, content management can also help capture the relevant information flows and turn them into an OSP knowledge base.

For example, in return for the free download of the software, an OSP could ask users to submit a user profile. The software would still be free of charge, but the users could contribute to the community by producing valuable information e.g. about their background, why they chose to download the product, and how they are planning to use it. This information would be useful for the software designers, content authors, as well as the information model, and thus ultimately the community itself. Filling in a user profile might either be a prerequisite for being able to download the software, or the last step of the installation if an installation wizard was used. If it was included as the last step of the installation, the form could also ask the user for feedback on the installation procedure and instructions. The request to submit a user profile should clearly indicate that all information will be handled confidentially and that it would only be used to improve the content provided by the OSP. The possibility to subscribe to a user forum could also be included as a step in filling the user profile. This approach would signal to the user how much the project values new contributions regardless of the individual's background. By storing the user profiles in a database, the data could be used e.g. to generate typical use cases, but also data that could be further distilled into diagrams and integrated to the contents of the OSP portal. It was already mentioned in 4.4.4 that the issue tracker provides useful information for project evaluators. Collecting user profiles represents another way of generating such information.

Furthermore, the possibility to fill in a contributor profile could be provided as a first step of contributing to the community. Collecting both user and contributor profiles would aid in building personalised content. By integrating the contributor profiles with other project tools and forums, the learning curve of a newbie contributor could be lowered significantly. As the OSP contributor would log in to the project web site, s/he could be automatically pointed to useful information resources, such as tutorials and/or development tasks that match her skills based on the profile that s/he committed. The contributions made could also be tracked and added to the profile, which would provide an easy way for the core developers to track and rate the contributor's productivity and skills. The following tables show the kind of data that could be collected in a

user/contributor profile. Creating a profile would mean filling in a form, and thus it would be very suitable to use structured authoring, e.g. XML.

Table 13. Metadata attributes for the “actor” dimension

Metadata attribute (dimension)	Value (subcategory)	Value (subcategory)
Actor	Evaluator, inactive user, reader	Background, language, scenarios of use
	Contributor	Role, background, area(s) of expertise, language

Table 14. Metadata attributes for the “contributor” dimension

Metadata attribute (dimension)	Value (subcategory)	Value (subcategory)	Value (subcategory)
Contributor	Role	Programmer, system administrator, project manager, information architect, content author, editor, translator, graphics designer, information designer, usability expert, tester,...	
	Background	Education, work experience	
	Area(s) of expertise	XML, XHTML, CSS, Java, Perl, Python, PHP, graphics design, usability,...	Skill level (beginner, advanced, expert)
	Language		
	Contributions		

Dimensions based on actors may not, however, suit all OSPs. When the project is targeted at a highly technical, homogeneous audience, the distinction between user and developer may be almost non-existent. In such a case, it may be more useful to use dimensions related to contributor role, skill level, or subject matter.

Subject-matter-based dimensions might be derived e.g. from the software architecture, functionality and modules. Furthermore, Oram (2006a) argues that "Some types of knowledge may be amenable to learning in dribs and drabs. Certain other subjects are deep and require a holistic approach." Areas of expertise that require a broad understanding consequently also deserve to be identified in the information model as a main dimension. Oram (2006a) gives the following as examples of such subject fields:

- Security - Security doesn't consist of installing all the patches from the vendor. It consists of an integrated approach to policies, risk assessment, and the disciplined monitoring of systems.
- Performance tuning - Few optimisations can be made in isolation. The term "tuning" is quite appropriate here, because performance is like trying to tune a keyboard instrument.(...) Tuning takes a sophisticated, nuanced approach--and different tunings are appropriate for different time periods and pieces.
- Troubleshooting - [...] You need a broad understanding of many different levels of a system to do problem-solving.
- Robust programming - In any programming language, bad habits are easy to fall into, and they come back to torment you later.

5.2 Information types and content units

The next step in creating an information model is to identify the relevant information types, each of which must be based on the needs of the user community (Hackos 2002, 40).

The three information types typically used in technical communication and end-user documentation and also the basic building blocks of DITA are reference, procedure and concept. Consequently, these types are certainly relevant for authoring OSS-related content as well. However, the discussion so far has also revealed that OSPs typically use other, distinguishable information types. These include frequently asked questions (FAQ), tutorials, software architecture descriptions, API reference, release notes, and issues (in issue tracking systems). The actor profiles suggested in the previous section are another example of a new information type.

Creating tutorials is important because of the main motivation for contributing: learning. FAQs, software and API descriptions lower the barrier of entry for new contributors. For the

same reason, the "issue" information type (especially that of a development task) should be planned carefully to make sure volunteers will be easily able to find suitable tasks. Release notes, on the other hand are common to all software products. However, there are no standards or general agreement about the content units required to construct a release note. Although release notes typically present a concise summary of the changes, enhancements, bug fixes as well as known bugs and possible workarounds related to a particular software release, there is typically significant variation between release notes produced by different authors/organisations. A standardised "release note" information type, together with a carefully designed "issue" information type for reporting and tracking software defects could help automize the generation of this information.

6 Functional requirements of open source content management

In this chapter, I will first look at content management from the point of view of designing and implementing a CMS. Again, there are many different approaches and models of implementation available to choose from. While I have chosen to use the content management strategy of Hackos (2002), I will at times present the views of other researchers or CMS experts to show which themes Hackos tends to emphasise, to clarify how she defines certain central terminology, and to ascertain that Hackos' model is aligned with the argumentation presented elsewhere in this study.

I will then relate Hackos' CMS framework to the second phase of the content management process and use all the information gathered thus far to summarise the functional requirements of open source content management. The functional requirements define:

- the required authoring and workflow capabilities
- the required storage, management, and retrieval functions of the CMS
- how the information needs to be assembled and delivered (Hackos 2002, 41).

Hackos emphasises that the functional requirements should only define the requirements, not specify or design possible solutions. Instead, after completing the functional requirements, it should be used as a request for proposal to ask vendors about possible content management solutions. This is a valid recommendation for OSPs, as the following commentary by Rothfuss (2002, 148) demonstrates:

> Many OSP[s] exhibit a[n] over [sic] fondness for tools and try to solve issues with tools that should be resolved with processes instead. Tools support processes, but cannot replace them. One pitfall is to set up dozens of data entry tools to gather user comments, bugs, status reports, summaries, documentation, and then wonder why no one finds anything anymore. It is often better to use fewer tools more extensively than many tools only superficially. The ease to setup new tools masks the real problems that only crop up later: duplicate information, data silos that gobble up information but never release it again.

Hackos (2002, 41-42) also recommends that each vendor candidate should provide a "proof of concept" for the final set of possible solutions. In the case of OSS, however, the functional requirements document might also play a different, more significant role. Consider what might

happen if all OSPs were to first create and then openly publish their information models and functional requirements to enable industry-wide review. As there are already several high-quality open source content management systems available (such as the Plone project[1]), I would assume that the open source CMS developers would be interested in having a look at the functional requirements of other OSS projects. If the information models and functional requirements were carefully prepared, they would be likely to provide valuable insights about the features and functionality that future CMSs should include. After all, as was pointed out in Chapter 2, content management is a relatively new industry, and its role in organisations is still evolving. In the Introduction, I set out to identify areas that would benefit from the close cooperation of programmers and non-programmers (e.g. software architects and information architects). Content management certainly represents a common ground where close interaction between experts with varied backgrounds could bear fruit. Or to coin the words of Raymond (2001), content management is a field where technical communication experts, usability experts, and computer science experts could do some symbiotic "itch-scratching".

6.1 Anatomy of a CMS

As Robertson (2003) points out: "[...] there are literally hundreds of content management systems, all having different capabilities and strengthts". Consequently, when it comes to choosing a CMS, there is no one-size-fits-all solution: as both Robertson (2003) and Hackos (2002, 40-41) emphasise, every organisation has a unique set of requirements that must be taken into account when planning for a CMS. Although they use slightly different wording, both Hackos and Robertson define the main areas of a CMS in the same way. The following figure, taken from Robertson (2003), shows the anatomy of a content management system. I have modified the text fields of the figure to show the phrasing used by Hackos in her definition of a CMS since I will use her guidelines as my theoretical framework to build an OSS information model.

1.http://plone.org/

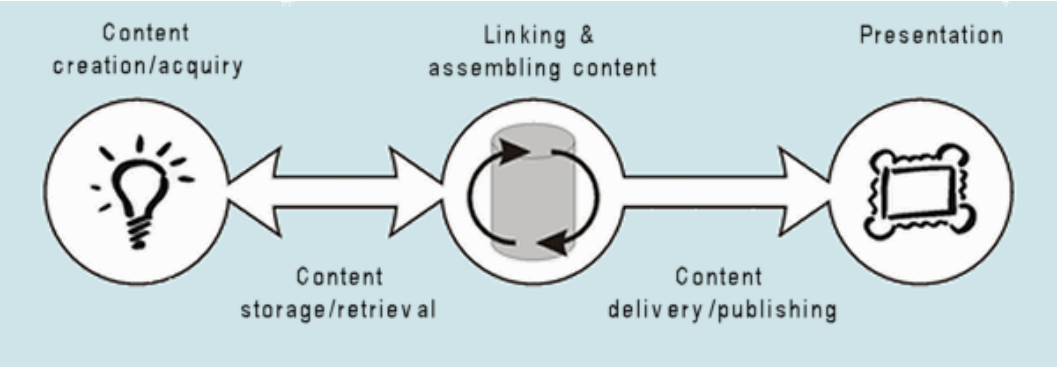

Figure 18. Anatomy of a CMS (Adapted from Robertson 2003)

As shown in Figure 18., a CMS consists of the following primary components:

- an environment for creating and acquiring content
- a repository for storing and retrieving content
- a method for assembling and linking content
- a delivery mechanism for delivering content to your customers (Hackos 2002, 52).

I will next describe each of these components in more detail to describe the various alternative features and functionality of CMSs and to later relate them to OSS-specific functional requirements. To follow Hackos' (2002, 52) advice, I will start at the end, that is, the audiences' needs and expectations regarding the delivered content.

6.1.1 Content assembly, output, and delivery requirements

An organisation must set up a system to design and deliver the right content and functionality in ways that meet the audience requirements (Boiko 2005, xliv). The demands of the output requirements also govern the input requirements, such as the editorial, metadata, and assembly requirements. Hackos (2002, 63) differentiates between two different assembly and delivery strategies:

- **Early binding**: the assembly and the delivery format of the information is determined before the content is checked into the content repository
- **Late binding**: The delivery format may be different from the format used to author the content. The content units are assembled when they are published from the repository. Styles and formatting are applied during the assembly and publishing process.

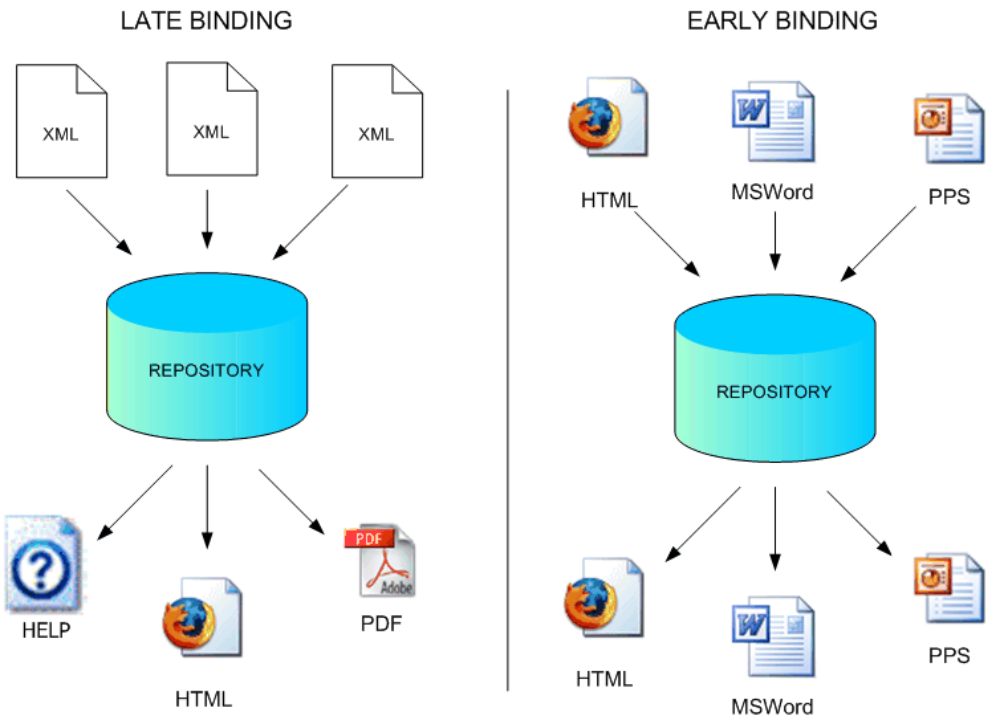

Figure 19. A comparison of late and early binding

I have visualised the late and early binding methods in Figure 19. based on Hackos' definition. In this example, late binding means that XML files are stored to the repository, and converted into help, HTML, and PDF files. In the early binding example, HTML, MSWord, and PowerPoint files are stored and published from the repository as-is.

Moreover, content assembly can be controlled by the authors, the information users, or automatically assembled for specific user groups (Hackos 2002, 94). Most technical information web sites are static, that is, the assembly is planned by the information architect and authors; the structure does not change according to user needs, and the web site is not updated dynamically as the information changes. The only typical exception to this rule is multiple language support, which allows readers to select a language e.g. from a list. (Hackos 2002, 259)

Technical information is typically organised according to:

- chronological order of tasks to be performed
- functional order related to the design of a product
- tasks that range from easy to difficult to perform
- conceptual order to best facilitate the learning process (Hackos 2002, 222).

A dynamic web site might also allow a user to choose between such alternative organisational architectures depending on his/her information needs, or even automatically organise the information according to pre-defined preferences, as the user logs in to the service.

It is still relatively rare for content to be delivered in dynamic form, as this is more difficult to design and implement. But dynamic delivery has great potential as it allows content to be personalised according to user needs. (Hackos 2002, 261-262). As Hackos (2002, 263-264) notes,

> "Both customi[s]ation and personali[s]ation support the goal of building communities of users. Communities are, of course, better served with information as their needs become better known. Communities are often eager to share information among colleagues, thereby enhancing the knowledge base. [...] In all, information designs that respond to user needs provide the starting point for community building."

Dynamic customisation, that is, assembling content on the fly from the content repository based on user needs in a particular circumstance (Hackos 2002, 267), is therefore very interesting from OSS community building point of view.

The required format, dynamics, structure, and organisation of the delivered content are determined in the information model. It may also be justifiable to construct different portions of a web site using different strategies ranging from static to dynamic content delivery and different methods of organisation.

6.1.2 Content repository, storage, and retrieval

The CMS of an organisation is normally a database, which is set up to store, categorise, and organise content and functionality outside of any particular delivery channel, thus making it easy to find and retrieve. (Boiko 2005, xliii) This basic functionality of a content repository is usually referred to as library services, which include functions such as check-in, check-out, and access and version control (Hackos 2002, 62). The basic functionality or technical requirements for a content repository are, however, not central to the scope of this study. Thus, I will move on to discuss more relevant aspects such as the suitable level of content granularity.

The correct size of the components or chunks of information to be stored in the repository depends on what is worth managing separately (Hackos 2002, 80). Again, it is the information model that provides the answers: the required level of granularity depends on the reuse strategy and the need for personalisation, single-sourcing and translating, among other things (Hackos 2002, 82). An organisation should define a single-sourcing strategy if it produces multiple versions of the same product, releases new versions frequently, on short notice, publishes information in multiple languages, or delivers the information in multiple media (Hackos 2002, 295). Hackos (2002, 304-306) distinguishes between the following levels of granularity:

- document-level granularity
- chapter-level granularity
- module or topic-level granularity
- content-unit-level granularity
- word-level granularity.

Some of the information resources of an organisation may be highly structured, while other information may be very free form. Nevertheless, each component that needs to be stored and retrieved as a separate component must be labeled and/or tagged with metadata. The CMS can use either

- file names in a flat-file system stored on file servers
- metadata attached to the files or objects stored in a database
- tags to identify the internal parts of each document or topic [...] (Hackos 2002, 61).

An untagged document can only be stored as a whole, without giving the authors or users information about what it contains unless they open the document (Hackos 2002, 68). For example, to gain the most from single-sourcing, the information should be stored at a finer level of granularity than whole documents. Consequently, there are many alternative approaches to choose from. Firstly, information can be stored as whole documents with metadata wrappers that identify the contents. Secondly, existing large documents or topics can be burst apart and the parts can be stored as components to be reused in other contexts. It is also possible to author individual

modules, identify them with metadata, and assemble them into compound documents or link them together to form a document, document set or a web site, for example. Lastly, content modules may also be structured into individual content units using XML tags to allow the search, retrieval, modification, linking, and assembly of individual components into other contexts. (Hackos 2002, 303-313) In some cases it may prove worthwhile to compromise and store some of the content at document level and others at topic level, for example. (Hackos 2002, 80)

6.1.3 Authoring and content-acquisition environment

The components of content for an organisation are produced in the authoring and content-acquisition environment. An organisation must set up editorial and metatorial systems to efficiently capture the information that needs to be delivered through the CMS:

- **editorial systems** ensure that the content has appropriate and consistent format and style
- **metatorial systems** ensure that the content is appropriately and consistently labeled and/or tagged. (Boiko 2005, xliii)

Consequently, a fundamental aspect of designing the content acquisition and authoring environment is deciding what information should be captured, i.e. included under content management and what should be excluded. This leads to questions such as the following:

- what information resources are most relevant or critical to the success of the organisation
- what content is used most frequently
- what content needs to be updated most frequently
- what information can simply be archived and how to handle the archiving.

Further investigation of the authoring and content-acquisition environment reveals that there often exists a great deal of diversity. This was, for example, the case with community authoring as explained in the previous chapter. Several different tools and modes of authoring may be used simultaneously in one authoring and content-acquisition environment, including:

- **unstructured authoring**: no templates or style tags are used (e.g. email)

- **forms-based authoring**: forms are completed in print or electronically with field-based authoring (e.g. issue tracking systems)
- **format-based authoring**: use of templates and style tags (MS Word, FrameMaker)
- **structured authoring**: content-based templates and content tags (XML and SGML) (Hackos 2002, 64).

Depending on the output requirements, a CMS may support only one authoring mode or even all of them.

There must also be an interface that associates the authoring and content-acquisition environment with the content repository and allows authors, editors, and translators to locate, reuse, modify, check in/out, assemble, and link the content components efficiently and easily (Hackos 2002, 93; 339). Consequently, the CMS must take into account aspects such as:

- required workflow and notification processes functioning between the authoring environment and the repository (for example, individual authors are notified if a particular content unit or topic is being modified by someone else)
- if the use of several repositories is required (for example, a project may need a publishing repository or a staging area in addition to the normal working repository: a robust publishing repository may be required in order to publish dynamic content while a staging area allows content to be tested before publishing)
- link management systems
- multiple language support (Hackos 2002, 90-95).

6.2 Summarising the functional requirements for open source

I will begin by repeating the prerequisites of open software development in order to translate them into requirements of open content acquisition and authoring and thus to answer the call of Berglund and Priestley (2001, 135) who argue that "open-source documentation requires a framework that captures the relevant qualities of open-source development (just-in-time and user-

driven development) while accommodating the special requirements of documentation development". The "just-in-time" and "user-driven" attributes are in effect derived from Raymond's (2001) remark of "scratching one's own itch": people develop what they want when they need it badly enough. According to Berglund and Priestley, the main goals of the framework are to allow people to contribute content and thus lower the barrier of entry and to turn technical debate (currently taking place in mailing lists, discussion forums, blogs, and planets) into formal support for open source software.

In chapter 3, it was concluded that openness of open source requires:

- modular software architecture meaning that the software is composed of components/functionality that interact with each other through clearly defined interfaces but are otherwise independent of each other
- ability to read the source code (i.e. the source code must be provided in a well-readable form) and to deduce the architecture/design from the source files
- ability to modify and experiment with the source files in parallel to other simultaneous development activities
- ability to extend, reuse, and redistribute software components/functionality
- open communication in a public space that allows anyone to follow the discussion and to participate in it
- ability to join the OSS community, and contribute at a freely chosen level of commitment.

Firstly, modularity is not an alien concept to documentation development or content management, either. As explained in chapter 3, modularity of content can be achieved by constructing information-type-specific topics, i.e. stand-alone chunks of information that do not require another topic to be understood. If a higher level of granularity is required, the information can be chunked into content-unit-level modules instead of topic-level modules. Modularity of content is also a prerequisite of single-sourcing i.e. reuse of content. Single-sourcing covers both repurposing and reassembly. Repurposing means delivering the same content in different output

formats, while the term reassembly refers to reorganising content units for different purposes or audiences. (Ament 2003, 15)

Secondly, the requirement to provide the source files in a well-readable form that allows readers to infer the underlying structure would strongly suggest the use of structured authoring i.e. XML or SGML. A DTD or an XML schema would be the most unambiguous way to show how the content units "interface" or relate to each other and how they can be used to build topics. Use of XML would also enforce the consistent use of the content components (i.e. topics and content units). Use of XML would also better enable redistribution of content, or, as Berglund and Priestley (2001, 135) put it, "the documentation source must be free to become part of many different projects". To enhance content reuse across open source projects, use of open standards and protocols such as Darwin Information Typing Architecture (DITA) would also be beneficial.

The second and third "requirements of openness" are in effect realised by the use of SCM tools, especially the basic functions of revision control. Similarly, parallel development of content units in turn can only be enabled by CMS library services, the equivalent of revision control in SCM.

Moreover, to achieve the ultimate level of openness in terms of "open product" and "open process" as defined by Ye and Kishida (2003) would mean that:

- All interim (i.e. draft) documentation versions would be available to all community members. Depending on the volume of the content base, the size of the community, and the amount of content contributors, a staging area might be required to allow testing and reviewing.
- The documentation design and development decisions and the argumentation behind these decisions would be conducted in a public space and everyone is free to join the discussion.

The argumentation presented above is, however, foremost a philosophical view of the open source content management requirements. Naturally, not all information resources of an OSP need to be written in XML or managed at content-unit level: as was observed earlier in section 6.1.2, it may

be better to compromise and use different strategies (e.g. dynamic vs. static delivery, methods of organisation, unstructured vs. structured authoring etc) to construct and deliver different pieces of content. The earlier discussion nevertheless reveal that it can be recommended to use structured authoring to create content that is typically included in traditional user's guides and standard operating procedures and guidelines. Furthermore, the DITA framework, which is especially designed for documenting software products and is also a open standard, would be ideal for established OSPs. Use of high-level granularity (e.g. content-unit-level granularity) to allow efficient single-sourcing would decrease the resources required for translating and also speed up the publishing process. As pointed out by Berglund & Priestley (2001, 132) one of the main dilemmas of OSS documentation is that the functionality of an open source product cannot be determined until the day it is completed. The continuous change of requirements makes traditional methods of software documentation difficult as writers cannot simply follow an existing documentation plan.

Within an OSS community, there are typically users with highly different backgrounds and skills. As was noted earlier in the study, a novice in one area may be an expert in another. Furthermore, OSS is used by different individuals use with different tools and even different operating systems. Use of Wikis and OpenOffice documents are one example of an attempt to overcome the related challenges, but, as discussed earlier in the study, Wikis and word processors typically do not impose the use of standardised information types and content units. Furthermore, although the surveys show that people are eager to contribute content to support the community, they are also quick to choose an authoring tool that allows them to do this quickly and conveniently. Instead of investing hours in studying potentially more advanced but often poorly-documented OSS documentation tools and processes, they may opt for documenting a troubleshooting trick in their own blog, thus subjecting fellow troubleshooters to the mercy of search engines such as Google. Such behaviour further adds to the fragmented nature of OSS-related content. Boiko (2005, 190) suggests that the content acquisition environment of a community web site should provide web

based forms, which guide contributors through the process of submitting small pieces of original content. The forms should also allow contributors to submit files and larger pieces of content, tag the files with metadata and upload them to the community repository. In my opinion, this applies especially to contributing product-related content for inactive users, readers, and contributors. An ideal solution would be a wiki-type interface, integrated with a DITA-based information model. Such a tool would provide a central point of access to the authoring environment and the content repository, an easy-to-learn syntax and user interface, and also enforce the use of standard metadata, information types, and content units. Consequently, both end-users and software developers who are not particularly interested in the finesses of creating technical information could focus on the contents, letting the authoring environment take care of the formatting, structure, organisation, and even linking. The words of Boiko (2005, 249) also suggest the validity of this argumentation:

> People will not naturally seek out the CMS and contribute to it. Rather, the CMS team must seek out contributors and go to them. To begin, you must define what makes information valuable to your organi[s]ation. If you do not know what is valuable, you will not know where to find it and will end up putting a lot of effort into information that is not important.

The fact that product-related content and project-related content for evaluators need to be translated, also demands efficient single-sourcing, a high-level of content granularity, and the use of XML. Guides for open source developer guides are typically provided only in English and are not translated.

Consequently, I believe that "plain" Wikis may be more suited for content targeted for (expert) developers: a developer has already become more committed to the project and may thus be more motivated to both design and follow authoring guidelines to ensure that the submitted content adheres to the guidelines.

OSPs already use web-based forms in issue tracking. Furthermore, issue tracking systems often already have the required features for managing issue-related content (that is, defects, tasks, enhancements, features, etc). What is typically missing is an information model that guides the

modification and personalisation of the issue tracker so that it covers the important dimensions and responds to the users information needs. As described earlier, issue trackers often allow integration with version control tools. They should also allow integration with content management systems to make it easier to both find suitable authoring tasks and to track their progress.

Boiko (2005, 190) also suggests that there should be batch-content processors that can collect information from mailing lists, chat sessions and label it along with other contributions, and e-mail acceptors that allow members to e-mail content directly to the community CMS. I believe that such an approach would work with simple questions and answers, up to a certain limit. However, as the amount of included questions and answers increases, so does the need for an information model that defines how the FAQs should be categorised to keep them them accessible and manageable. A more useful solution might to define a web-based form for the information type "FAQ". Whenever a user submitted a FAQ, the system could generate an email to the relevant user/developer forums.

To summarise the findings based on the discussion thus far, it is interesting to consider the recommendations Boiko gives for designing the CMS of a community site. According to Boiko (2005, 189-190), the CMS must:

- present relevant, targeted, personalised information to community members
- provide a "strong but not overly complex metadata framework that naturally guides members to actively contribute relevant, well-tagged information" to the community knowledge base
- include a workflow system for routing contributions through editorial and metatorial processing
- include semistructured sources coming out of a message center, for example, include e-mail threads and messages that can be made into topics
- provide a repository with "a fine level of categorisation to support maximum

personali[s]ation"

- enable the repository to grow in a constrained way so that content expires when necessary, missing or ambiguous information is easily identifiable, and new content areas are quickly ready for publication.

As we can see, the advice given by Boiko repeats several of the requirements suggested on the basis of the analysis.

The last, but definitely not the least important requirement is an advanced search mechanism that would allow an information search to be defined based on a variety of metadata dimensions. The search function should also allow the user to select the media (online forums, wiki, official documentation, issue tracker, etc) that the search is extended to.

6.2.1 Content acquisition and authoring requirements for open source

Access to the information model.

Assessing authoring requirements: at a minimum the same as in SCM.

Workflow:

- generating email notifications
    - content commits in addition to code commits. Need to distinguish between content modification and editorial changes: who should receive the email?
    - issue tracking systems: the notification could be sent to specific mailing lists based on the attributes (i.e. metadata) selected for the issue.

Ranking systems, review process, multiple dev platforms, authoring tools, and file types (graphics)!

Security updates - specific procedure/workflow

The different modes of authoring coincide with the way Boiko (2005, 89-91) distinguishes between acquired and authored information: acquired information means information that was not originally created for the CMS in question but has been acquired as syndications or found

sources. Syndications are "sources that are designed for reuse" (i.e. it is delivered in a reusable format such as XML) while found sources are files that "you have come across or have been given to you" and which must therefore be processed before adding them to the CMS. Authored information is typically low volume but high quality, while acquired information is high volume but low quality.

Andy Oram:

- online training
- funding and rating systems
- search facilities.

7 Conclusions

The main purpose of this study was to assess if open source documentation does not fulfill the requirement of openness the way open source does. To provide a framework for the comparison, a conceptual analysis was first used. Later in the study, other ways of defining the requirement of openness were presented based on existing OSS-related research.

The conceptual analysis showed that information (e.g. a document) is by nature less concrete, reusable, and manageable and more ambiguous than data (e.g. code). Consequently, information can be considered to be less "open" by nature than code. The conceptual analysis also showed that open source code typically consists of functionality that has been packaged in objects or in blocks of programming code. Such functionality can be categorised as content. Furthermore, information can also be transformed or "datatised" into content by adding semantic metadata that makes explicit the intended context, connotation, and interpretation originally intended by the composer of the information. Transforming information into content by using metadata that has been identified in an OSS-community-based information model therefore represents a solution that can be used to overcome the obstacle of "closedness" presented by the differences in the inherent characteristics of information and data.

An analysis of OSS-related research was used to identify other definitions of "openness" of open source. The discussion showed that requirement of "openness" can mean:

- Openness of the OSP's processes, which aim to lower the barriers of entry: an OSP must be open to join by new volunteers wanting to contribute and also open to the volunteers' choice of work.
- Openness of the product or deliverables and the corresponding development activities.
- Modular architecture design.
- Open communications, management, and infrastructure.

It has also been shown that in most cases the existing research on how these factors are manifested in actual OSPs only study the openness from the aspects of programming and

software development. Very little if any attention is paid to peripheral participation such as documentation or community authoring, the importance of such contributions, or the background of the individuals making these contributions.

In the Introduction, I presented the hypothesis that open source development and the development of open content are similar to a great extent. The process of creating open source code and content was first discussed at an abstract level, in terms of the continuum of understanding. The conceptual analysis showed that in their work, both a programmer and an information architect use their intellect and knowledge and disintegrate it into discrete pieces of data, functionality, and content in an attempt to produce useful, high-quality open source software products or publications. Furthermore, a comparison of OSS design and a content management system showed that these two systems have several common prerequisites including modularity of architecture, reuse or single-sourcing, strict definition of interfaces and interrelations, and linking of components, and a need for revision control and parallel development.

The architecture of open source software is often thought to be discernible from the source code. However, the needs assessment phase, which included an inventory of the OSS information resources and an analysis of the user experience regarding the available information, showed that simply delivering the code in a well-readable form is usually not sufficient to achieve the requirements of "openness". New volunteers often need additional software architecture and API descriptions to aid them in understanding the architecture and design and making useful contributions. The requirement to deliver the OSP and product-related content in well-readable form demands the use of structured authoring based on semantic metadata and information typing and/or access to the information model because the information architecture is not easily discernible from unstructured documents.

As the last step of the needs assessment phase, ideas about the benefits of open information models were presented. The standardisation of metadata leading to more efficient reuse and

improved usability and accessibility of OSS-related information was identified as the most important benefit of open information models.

A general-level information model was also presented, and actor groups and their profiles and the tasks of learning and contributing were identified as the most important dimensions of the information model.

Several suggestions for future development arouse during the making of this study. As mentioned in the Introduction, very little if any research could be found that would have focused on documentation or content-management-related aspects of OSS development. Although the possibility to perform a case study would have benefited the discussion greatly, I was forced to discard such an approach because during the research it became apparent that a general framework must first be established. Consequently, this study became more focused on the needs assessment phase, while the information model and functional requirements of an open source CMS remain at a very general level. Case studies of interrelated projects or projects from the same industry would be likely to produce more detailed data for information modeling and the functional requirements, and thus also provide more evidence e.g. about the efficiencies that might be achieved with metadata standardisation and single-sourcing of content across OSPs. It would also be of interest to investigate the openness of documentation and/or content management in specific projects using the framework established in this study.

Issue tracking systems were recognised as a content management system of their own, which also require more study. There are many interrelated dimensions related to issue management, and therefore the issue tracking system of an OSP would rightfully deserve an information model of its own.

The authoring community would also deserve more research. The OSS user/developer surveys done so far have approached the contributors from a software-focused point of view. An OSS-related survey, performed from an author or content management point of view might produce some interesting insight about the efficiency of content management workflows in OSPs, the

value of peripheral contributions and contributors to the success of OSPs, as well as the variety of roles played by individual OSS contributors, and the power relations between OSP members.

Works cited

Ament, Kurt 2003. *Single Sourcing. Building Modular Documentation*. Norwich: William Andrew Publishing.

Berdou, Evangelia 2007. *Managing the Bazaar: Commercialization and peripheral participation in mature, community-led Free/Open source software projects*. London: London School of Economics and Political Science. [Internet]. Available from: <http://opensource.mit.edu/papers/PhD_Berdou.pdf> [Accessed 03 March 2008]

Berglund, Erik and Priestley, Michael 2001. *Open-Source Documentation: In Search of User-Driven, Just-in-Time Writing*. In Proceedings of the 1999 International Conference on Software Engineering, 132-141. Santa Fe, NM, USA: ACM Special Interest Group for Design of Communications. [Internet] Available from: <http://xml.coverpages.org/DITA-Berglund2001.pdf> [Accessed 23 January 2006]

Bleek, Wolf-Gideon and Finck, Matthias 2004. *Migrating a development project to open source software development*. In *Collaboration, Conflict and Control - Proceedings of the 4th Workshop on Open Source Software Engineering*, ed. Joseph Feller, Brian Fitzgerald, Scott Hissam, and Karim Lakhani, 9-13. Edinburgh, Scotland. [Internet] Available from: <http://opensource.ucc.ie/icse2004/Workshop_on_OSS_Engineering_2004.pdf> [Accessed 03 March 2008]

Boiko, Bob 2005. *Content Management Bible*. Indianapolis: Wiley.

Carlson, Scott 2006. *Wanted: Female Computer-Science Students*. In *The Chronicle of Higher Education*. [Internet] Available from: <http://chronicle.com/free/v52/i19/19a03501.htm> [Accessed 3 May 2008]

Clark, Donald 2004. *The Continuum of Understanding*. [Internet]. Available from: <http://www.nwlink.com/~donclark/performance/understanding.html> [Accessed 10 March 2008]

CM3 2008. *What is a content management system*. [Internet]. Available from: <http://www.cm3cms.com/company/articles/whatisit.html> [Accessed 03 March 2008]

Fogel, Karl 2005. *Producing Open Source Software: How to Run a Successful Free Software Project*. [Internet] O'Reilly Media. Available from: <http://producingoss.com/> [Accessed 03 April 2008]

Ghosh, Rishab Ayer 2004. *The Social Context of Free Software* [Internet] Available from: <http://www.infonomics.nl/FLOSS/papers/20040509/50.htm> [Accessed 03 April 2008]

Hackos, Joann T. 2002. *Content Management for Dynamic Web Delivery*. New York: John Wiley & Sons.

Helander, Nina and Laine, Jarkko 2006. *The Value Network Approach to Open Source Software Business*. In *Multidisciplinary Views to Open Source Software Business*, ed. Nina Helander and Hanna Martin-Vahvanen, 46-57. Tampere: Tampere University of Technology and University of Tampere. [Internet]. Available from: <http://www.ebrc.info/kuvat/eBRC_rr33.pdf> [Accessed 21 January 2008]

Helander, Nina and Antikainen, Maria 2006. In *Essays on OSS Practices and Sustainability*. Tampere: Tampere University of Technology and University of Tampere. [Internet] Available from: <http://www.ebrc.info/kuvat/eBRC_rr36.pdf >[Accessed 27 January 2008]

Hey, Jonathan 2004. *The Data, Information, Knowledge, Wisdom Chain: The Metaphorical Link*. [Internet] Available from: <http://www.oceanteacher.org/oceanteacher/index.php?module=contextview&action=contextdownload&id=gen11Srv32Nme37_1590> [Accessed 03 March 2008]

Jones, M.T. 2006. *Version control for Linux*. [Internet] IBM. Available from: <http://www.ibm.com/developerworks/linux/library/l-vercon/> [Accessed 16 April 2008]

Kennedy, Patrick 2007. *The many faces of information architecture*. [Internet] Step Two Designs Pty Ltd. Available from: <http://www.steptwo.com.au/papers/kmc_iafaces/index.html>[Accessed 03 March 2008]

Linux Documentation Project 2006. *Documentation Definition*. [Internet] Available from: <http://www.linfo.org/documentation.html> [Accessed 20 March 2008]

Luoma, Ilkka 2006. *Software Engineering in Open Source Software*. In *Essays on OSS Practises and Sustainability*, ed. Nina Helander and Maria Antikainen, 56-69. Tampere: Tampere University of Technology and University of Tampere. [Internet] Available from: <http://www.ebrc.info/kuvat/eBRC_rr36.pdf > [Accessed 27 January 2008]

Markova, Maiju 2005. *Tiedon merkitys organisaation muuttumiselle ja uudistumiselle*. Tampere: Tampere University of Technology and University of Tampere. [Internet] Available from: <http://www.ebrc.info/kuvat/eBRC_RR27.pdf> [Accessed 21 January 2008]

Matuska, Martin 2003. *Kategorisierung von Open Source Projekten: Aufbau- und Ablauforganisation*. Wien: Wirtschaftsuniversität Wien. [Internet] Available from: <http://pascal.case.unibz.it/retrieve/2673/matuska.pdf> [Accessed 21 March 2008]

Mikkonen et al. 2006. *Survey on four oss communities: Description, analysis and typology*. In *Empirical Insights on Open Source Software Business*, ed. Nina Helander and Maria Mäntymäki, 52-66. Tampere: Tampere University of Technology and University of Tampere. [Internet]. Available from: <http://www.ebrc.info/kuvat/eBRC_RR34.pdf> [Accessed 21 January 2008]

Miller, F.J. 2002. *I = 0 (Information has no intrinsic meaning)* In *Information Research*, 8, 1, ed. T.D. Wilson. [Internet] Available from: <http://informationr.net/ir/8-1/paper140.html#sve97> [Accessed 10 March 2008]

Mitsubishi Research Institute Inc 2004. *Free/Libre/Open Source Software Asian Developers Online Survey (FLOSS-ASIA)* [Internet] <http://oss.mri.co.jp/floss-asia/summary_en.html> [Accessed 03 April 2008]

Mork, Håvard 2006. *Documentation Practices in Open Source - A Study of Apache Derby*. Trondheim: Norwegian University of Science and Technology. [Internet] Available from: <http://hmork.mine.nu/articles/Documentation_Practices_in_OSS.pdf> [Accessed 03 January 2008]

Morville, Peter and Rosenfeld, Louis 2007. *Information Architecture for the World Wide Web.* Sebastopol, CA: O'Reilly.

Open Source Initiative 2006. *The Open Source Definition*. [Internet] Available from: <http://www.opensource.org/docs/osd> [Accessed 10 March 2008]

Oram, Andy 2004. *Splitting Books Open: Trends in Traditional and Online Technical Communication*.[Internet] Available from: <http://www.oreillynet.com/pub/a/oreilly/opensource/news/2004/09/23/online_trends.html > [Accessed 03 January 2008]

Oram, Andy 2006a. *Rethinking Community Documentation.*[Internet] Available from: <http://www.onlamp.com/pub/a/onlamp/2006/07/06/rethinking-community-documentation.html> [Accessed 03 January 2008]

Oram Andy 2006b. *Do-It-Yourself Documentation? Research Into the Effectiveness of Mailing Lists.* [Internet] Available from: <http://praxagora.com/andyo/professional/mailing_list/mailing_list.html> [Accessed 03 January 2008]

Oram, Andy 2007. *Why Do People Write Free Documentation? Results of a Survey.* [Internet] Available from: <http://www.onlamp.com/lpt/a/7062> [Accessed 03 January 2008]

Raymond, Eric Steven 2001. *The Cathedral & the Bazaar : Musings on Linux and Open Source by an Accidental Revolutionary*. Sebastopol (CA): O'Reilly. [Internet] Available from: <http://safari.oreilly.com/0596001088> [Accessed 03 March 2008]

Robertson, James 2003. *So, what is a content management system?* [Internet] Step Two Designs Pty Ltd. Available from: <http://www.steptwo.com.au/papers/kmc_what/index.html> [Accessed 03 March 2008]

Robertson, James 2004a. *Definition of information management terms.* [Internet] Step Two Designs Pty Ltd. Available from: <http://www.steptwo.com.au/papers/cmb_definition/index.html> [Accessed 01 March 2006]

Robertson, James 2004b. *Open-source content management systems.* [Internet] Step Two Designs Pty Ltd. Available from: <http://www.steptwo.com.au/papers/kmc_opensource/index.html> [Accessed 03 March 2008]

Robertson, James 2005. *10 Principles of effective information management*. [Internet] Step Two Designs Pty Ltd. Available from: <http://www.steptwo.com.au/papers/kmc_effectiveim/index.html> [Accessed 03 March 2008]

Rothfuss, Gregor J. 2002. *A Framework for Open Source Projects.* Zurich: University of Zurich. [Internet] Available from: <http://greg.abstrakt.ch/docs/OSP_framework.pdf> [Accessed 03 January 2008]

Salvo, Michael J. 2004. *Rhetorical Action in Professional Space: Information Architecture as Critical Practice.* In *Journal of Business and Technical Communication* 18, 1, 39-66. Sage Publications.

Sowe S.K. et al. 2008. *Understanding knowledge sharing activities in free/open source software projects: An empirical study.* In *Journal of Systems and Software* 81, 3, 431-446.

Stürmer, Matthias 2005. *Open Source Community Building.* Bern: University of Bern. [Internet] Available from: <http://opensource.mit.edu/papers/sturmer.pdf > [Accessed 21 January 2008]

Tyler, Chris 2006. *Documentation: A Key to Openness* [Internet] Available from: <http://epresence.senecac.on.ca/archives/2007_apr13_633120663020312500/?archiveID=20> [Accessed 16 January 2008]

Vainio, Niklas and Vadén, Tere 2006. *Sociology of Free and Open Source Software Communities: Motivations and Structures.* In *Multidisciplinary Views to Open Source Software Business.* ed. Nina Helander and Hanna Martin-Vahvanen, 10-22. Tampere: Tampere University of Technology and University of Tampere. [Internet] Available from: <http://www.ebrc.info/kuvat/eBRC_RR33.pdf> [Accessed 21 January 2008]

Vainio et al 2006. *Elements of Open Source Community Sustainability.* In *Essays on OSS Practises and Sustainability*, ed. Nina Helander and Maria Antikainen, 4-31. Tampere: Tampere University of Technology and University of Tampere. [Internet] Available from: <http://www.ebrc.info/kuvat/eBRC_rr36.pdf > [Accessed 27 January 2008]

Vixie, Paul 1999. *Software engineering.* In *Open Sources: Voices from the Open Source Revolution*, ed. Chris DiBona, Sam Ockman, and Mark Stone, 91-100. Sebastopol, CA: O'Reilly. [Internet] Available from: <http://www.oreilly.com/catalog/opensources/book/vixie.html> [Accessed 21 January 2008]

Wilson, T.D., 2002. *The nonsense of 'knowledge management'*. In I*nformation Research*, 8, 1, ed. T.D. Wilson. [Internet] Available from: <http://informationr.net/ir/8-1/paper144.html> [Accessed 4 March 2008]

Ye, Yunwen and Kishida, Kouichi 2003. *Toward an Understanding of the Motivation of Open Source Software Developers.* Proceedings of the 25th International Conference on Software Engineering, 419-429. Portland, Oregon. [Internet] Available from: <http://l3d.cs.colorado.edu/~yunwen/papers/ICSE03.pdf> [Accessed 28 January 2008]