NTNU
Innovation and Creativity

OTS-Wiki: A Web Community for Fostering Evaluation and Selection of Off-The-Shelf Software Components

Kristian Aaslund
Simon Larsen

Master of Science in Computer Science
Submission date: June 2007
Supervisor: Reidar Conradi, IDI
Co-supervisor: Carl-Fredrik Sørensen, IDI
Claudia Patricia Ayala Martinez, Technical University of Catalunya

Norwegian University of Science and Technology
Department of Computer and Information Science

Problem Description

The goal of this thesis is to start the implementation of OTS-Wiki, a web portal for fostering evaluation and selection of Off-The-Shelf software components. The portal will provide users with wiki-style collaboration to incrementally populate the information database. Both subjective and objective information is needed to help users selecting the right component. OTS-Wiki will address the problems and disadvantages of related OTS web portals and provide a usable starting point for further development and research.

Assignment given: 20. January 2007
Supervisor: Reidar Conradi, IDI

OTS-Wiki: A Web Community for Fostering Evaluation and Selection of Off-The-Shelf Software Components

Kristian Aaslund
Simon Larsen
{aaslund, simonl}@stud.ntnu.no

Supervisors: Carl-Fredrik Sørensen and Reidar Conradi
Co-advisor: Claudia Patricia Ayala Martinez

Department of Computer and Information Science

NTNU
Norwegian University of Science and Technology

Abstract

Many challenges arise when it comes to selection of Off-The-Shelf (OTS) software components to use in software development. To make the selection process easier, other users' experiences on working out similar tasks could be of great value. Such experience data concerning the selection, the use, and the integration of a component is often not available.

In this Master thesis, we describe the implementation of OTS-Wiki, a wiki based portal with community driven content. OTS-Wiki is a web community fostering evaluation and selection of OTS software components. The wiki provides basic functionality to support the process of evaluating and selecting components for use in component-based software development.

The success of OTS-Wiki relies heavily on the quality and relevance of the content populating the repository. There are also usually related heavy start-up costs to such repositories. A wiki based portal, based on the open source collaboration principle where the users themselves controls and populates the repository could be the solution to both these issues.

Keywords: Community Driven, Component Evaluation, Component Selection, Off-The-Shelf, Open Source, OTS, OTS-Wiki, Social Computing, Web Communities, Web Technology, Wiki

Preface

This report is a result of the Master thesis project titled *OTS-Wiki: A Web Community for Fostering Evaluation and Selection of Off-The-Shelf Software Components*. The project is carried out as a co-operation project between the students Kristian Aaslund and Simon Larsen at the Department of Computer and Information Science (IDI) at the Norwegian University of Science and Technology (NTNU).

We would like to thank our supervisors Dr. Carl-Fredrik Sørensen and Professor Reidar Conradi, and our co-advisor Claudia Patricia Ayala Martinez for good advices through the project period. A special thanks to Carl-Fredrik Sørensen who has given us valuable feedback and guidance at our weekly meetings, and to Per Kristian Schanke for server hosting and administration.

Trondheim, June 2007

Kristian Aaslund Simon Larsen

Contents

List of Tables

List of Figures

Part I

Introduction

Chapter 1

Introduction

1.1 Background and Motivation

The use of prefabricated, off-the-shelf (OTS) software components has grown rapidly, especially in large-scale projects [Li et al., 2004]. OTS components are typically divided into two categories: *Commercial Off-The-Shelf* (COTS) produced and sold by commercial vendors, and *Open Source Software* (OSS) components provided by open source communities and made freely available for reuse and further development.

Traditionally, software vendors have produced most of their software in-house, possibly making entire systems from scratch. Over the years, software reuse has become an increasingly important "best practice". Making pieces of software (components) that easily can be reused in other projects is a time-saving and effective means of developing software systems.

Using OTS components may introduce a number of additional risks and challenges in software projects [Li et al., 2005]. As *Component-Based Software Engineering* (CBSE) grows in popularity, the number of available OTS, both COTS and OSS components is ever increasing. Identifying areas where an OTS component may benefit a project is perhaps the easiest part, and a good place to start. Selecting the right component in a vast number of available OTS may be far more difficult. Storing the knowledge and experiences of OTS projects is important to support systematic use and reuse of components for future projects. Also, complete and high-quality data concerning the available OTS is crucial to select the right component.

A major obstacle for CBSE is the lack of available, efficient, and quality-assuring means of searching, selecting, and evaluating OTS components. Many organizations fail when using OTS components due to high start-up costs of populating a repository of reusable information and inadequate tools for storing the OTS knowledge and lessons learned.

To meet these challenges, [Ayala et al., 2007] propose the development of an open wiki-based portal for sharing and reusing OTS information. The portal, named **OTS-Wiki**, has three main goals:

G1 *Fostering an OTS Community and Incremental Population of Content*

G2 *Federating Actual Efforts for Locating and Selecting OTS Components*

G3 *Enabling Systematic Support for Selecting and Evaluating OTS Components*

OTS-Wiki is based on *open source like collaboration* to encourage the OTS community to share both subjective and objective information about available components, supporting the processes of selecting, using, and evaluating OTS components.

1.2 The Goal of the Thesis

This thesis serves as an extention of the work of [Ayala et al., 2007], aiming to start the implementation of OTS-Wiki, evaluate the work in progress, and suggest changes and enhancements to the portal. The effort needed to develop and deploy a complete portal, covering all the visions and goals of OTS-Wiki, is beyond the scope of this thesis. The main goals is to develop a starting point to test the ideas and suggestions in practice, and to be able to evaluate the choices made. This will hopefully provide a valuable experience for further work on the OTS-Wiki and the related research.

1.3 Research Questions

The thesis will address the following research questions:

RQ1 What OTS tools and communities exist today, and what are their advantages and disadvantages?

RQ2 How can OTS-Wiki contribute to the OTS community?

RQ3 How can OTS-Wiki be implemented?

- **RQ3.1** *What technologies can be used to develop OTS-Wiki?*
- **RQ3.2** *What are the key requirements of OTS-Wiki?*
- **RQ3.3** *How well does the implemented system support usability?*

1.4 Contributions

This is the list of the contributions presented in this thesis. Table 1.1 shows which research questions each contribution conforms to, and where to find them in the report.

C1 A description of work and projects related to the OTS community

C2 A vision for OTS-Wiki and potential user scenarios

C3 An initial implementation of OTS-Wiki, enabling usability testing and further development

- **C3.1** *A description of Java as an enabling technology for developing OTS-Wiki*
- **C3.2** *A requirements specification describing both the functional and the non-functional requirements of OTS-Wiki*
- **C3.3** *An overall architecture supporting the development of a web portal*
- **C3.4** *An implementation evaluation describing what are actually implemented, and how*
- **C3.5** *A usability evaluation describing usability in OTS-Wiki*

Table 1.1: Contributions vs Research Questions

Research Questions	Contributions	Chapters/Sections
RQ1	C1	Chapter 4
RQ2	C2	Chapters 2, 8, and 9
RQ3	C3 and C3.4	Chapters 5, 6, and 7 and Section 8.1
RQ3.1	C3.1	Section 5.4
RQ3.2	C3.2 and C3.3	Chapters 6, and 7
RQ3.3	C3.5	Section 8.2

1.5 Report Outline

This report is divided into five parts. Part 1 consists of this introduction, with background information, and the problem definition. Also included in this part are the research questions and an overview of the contributions made in this thesis. Lastly, we present the OTS-Wiki vision and describe some potential user scenarios.

Part 2 covers the project's prestudy and presents the state-of-the-art of component-based development, open source software, social computing, and agile software development processes. This part also describes related work and presents some important challenges and issues related projects have discovered.

Part 3 presents the OTS-Wiki, a web portal system developed to foster evaluation and selection of off-the-shelf software components. We outline the system and its purpose, and describe the development process used. The main contribution in Part 3 is the system specification and software architecture. Here, we describe the system's requirements, structure, and design.

Part 4 documents the evaluation work of the project. We evaluate the system's usability and functionality to answer the research questions and decide the future of the OTS-Wiki. We also evaluate the development process and criticize the work done and choices made.

Part 5 presents the conclusion and recommendations for further work.

Chapter 2

The OTS-Wiki Vision

2.1 The Vision

The main vision of OTS-Wiki is to foster evaluation and selection of off-the-shelf software components. The system is a web-based portal, using the open Wiki principle with accessible and editable data relevant for the OTS community. Implementing the entire vision of the portal far exceeds the scope of this thesis, but ideas and plans are important to understand why the OTS-Wiki is developed, and to guide the further development.

An important feature is the collection and presentation of both subjective and objective information. The basic information about a component, such as name, description, and hyperlink, as well as which categories, technologies, platforms, and tags the component belongs to, is regarded objective information. The evaluation part of the portal is where the subjective information becomes important. An evaluation of a component can range from a simple rating, like a diceroll or a grade 1-10, to a more comprehensive evaluation describing separate parts of the component in detail. The important part is making both the objective and the subjective data easy to enter, and equally easy to retrieve and use.

Below, we will discuss some of the important ideas and features of OTS-Wiki.

2.1.1 Component Metadata

OTS-Wiki will consist of information about a large number of different components. This information has to be structured and easily accessible to the users. To achieve this, we suggest establishing a schema of metadata attributes suitable of classifying the components. The attributes are grouped into five categories:

Domain Function
: What domain the component belongs to, and what function it has in that domain. This may form the basis of a set of categories and subcategories the component can be placed in. A component may belong to one or more category.

Quality
: The quality attributes of the component, based on the ISO 9126 standard for evaluation of software [Jaccheri and Torchiano, 2002, Torchiano et al., 2002]. This may include

objective information, such as reported reliability, computed complexity, functionality, efficiency, maintainability, and portability.

Technical Properties
Properties, such as *technologies* and *programming languages* used, *platform*, *size* (Source Lines of code (SLOC)), *software standard*, and *architectural patterns*.

Non-Technical Properties
Properties, such as *name*, *description*, *website*, *vendor*, *versions*, *dependencies on other components/software*, *licenses*, *documentation*, and *history* (when changed and by whom).

Experience Properties
This is the evaluation part of the portal, and may include *ratings*, *comments*, *experience reports*, *number of downloads*, *development activity*, *error rates*, *reputation*, and *future prognosis*.

2.1.2 Exchanging Experience

Many actors in the OTS community hold experiences and knowledge about certain components, as users of the components, as developers of the components, or as other component stakeholders. The experiences of these actors may play an important role in guiding OTS-Wiki users during the OTS selection process. A user can be registered as a resource person for a component, or a set of components. The experiences of this "*component uncle*" is made available alongside the component for reference to interested users. The evaluations performed by a component uncle may also be marked as more relevant because of the knowledge of this person.

Consider the following analogy:
A prospective student at NTNU is wondering which graduate study to apply to. He has two sources of information: The official information listed at the university's webpage, and hands-on experiences from a current or former student. In most cases, the experiences are far more valuable than statistics and official statements.

A person's experiences and impressions are often tacit information. The challenge lies in making this information explicit and available to others. Solving this challenge may significantly enhance the selection process in OTS-Wiki.

2.1.3 Using the Wiki Principle

The wiki principle is one of the main ideas behind OTS-Wiki [Ayala et al., 2007]. Enabling the OTS community to collaboratively and incrementally build and maintain an open database of OTS components will hopefully facilitate a growing and active community of users. Making the content open and accessible to all, without the need to register, is an important principle in OTS-Wiki. To make contributions, such as adding, editing, and evaluating the components, registration is however required. This is done to ensure a certain degree of quality on the content provided, and will enable users to sign the contributions, making the changes and contributors more visible.

2.1.4 User Profiling

There exist OTS components for a large number of different purposes and tasks. This is also reflected in the great diversity of the potential users of OTS-Wiki. User profiling is important to serve the different needs of the users. User profiling may be implemented both explicit and implicit. The explicit user profile is the information the user explicitly provides, such as demographical information, interests, job and educational information, and other. The implicit information is perhaps the most interesting source of user profiling. It includes user activity in the system, popular components or searches, or specific patterns. This information must be gathered automatically by the system. Implicit information from one or a few users might not be enough to identify popular activities or patterns in the use of the system. When the information source grows, patterns, and even bottlenecks may identify themselves.

One way of collecting the implicit information is to use logging. Logging can be applied at multiple levels in an application. At source code level, logging can occur at specific points in the program to report errors or usage of specific functions. On a higher level, logging can be applied to collect statistics about clicks and visitor counts. This is typically implemented by a web container.

Collecting relevant information about the users and their activities may enhance the portal, both with usability and efficiency in mind. Shortcuts can be provided to popular components or searches. Also, contributing users may earn higher status and be granted more rights in the system based on the information provided. This increases the quality of the data in the database, as posters of relevant information are rewarded, and "garbage posters" are ignored or banned. This is known as a "gift economy".

2.1.5 Web-Intelligence

Web-intelligence techniques may be used to analyze and extract actionable meaning from both structured and unstructured data (mostly textual), e.g., from user-provided dialogs, comments and ratings, experience and test reports, queries and query responses, actual component choices etc. Such an advanced knowledge system will exploit re-users' computer-tractable knowledge, relying on a web protocol to inter-work with other systems, e.g., automatic reasoning engines, ontology builders, classifiers, crawlers, search engines, even spy tools etc. This kind of open tool architecture effectively facilitates decomposition of the proposed OTS-Wiki software into independent pieces, i.e., constituting a Decision Support System with both new and existing knowledge-processing tools. The techniques may be used to automatically populate and update the portal's data from the component's webpage, and to perform intelligent searches both internally and externally.

2.1.6 Open Source Project

A part of the vision for OTS-Wiki is to make the project available as open source and enable further development by the OSS community. We believe that the open and collaborative approach will benefit the future of OTS-Wiki significantly. Encouraging the portal users and other developers to contribute to the portals's evolution creates ownership to the portal and the community. This will hopefully increase the quality and the level of functionality of OTS-Wiki beyond what can be achieved in a closed environment within the two participating universities.

An important task in enabling the OSS community to contribute to the system, is to provide suitable tools and infrastructure to support open source collaboration. Table 2.1 presents some

suitable tools. The task of making the project available as open source is beyond the scope of this thesis, but some of the work will be performed by a related project[1].

Table 2.1: Tools enabling OSS collaboration

Tool	Concern	Examples
Version control system	Manage revisions and evolution of the source code	Subversion, CVS, CVSNT, OpenCVS, GNU Arch, ArX, Monotone
Bug and issue tracker	Register and follow-up bugs, issues, improvements, and other changes in a structured way	Bugzilla, JIRA, Request Tracker, Roundup, itracker, Argus
Project management tool	Manage projects and provide a platform for collaboration	activeCollab, TWiki, MediaWiki, Open-Xchange
Open Services Gateway initiative (OSGi) framework	Tools to manage an application's life cycle, service registry, and modularity	Knopflerfish, Equinox, Apache Felix, Spring-OSGi

2.2 Scenarios

What can the OTS-Wiki portal be used for? We suggest a user scenario for potential usage of the portal. Also, we present two concrete user scenarios presented in a research proposal (Community Collaboration and Web-Intelligence to Support Development with Off-The-Shelf Software Components [NTNU, 2007]), using OTS-Wiki as an example, *EurOTS* and *IT consultants for SMEs*.

2.2.1 The OTS Developer

Consider the work of a software developer in an organisation that has just started to use OTS components in their work. Neither the developer nor the organisation has much knowledge about component-based software engineering (CBSE). A set of requirements has been identified as suitable for OTS adoptation, but no-one knows which components to use, or how to use them.

The developer is put in charge of selecting the component or components that best match the wanted requirements. As a beginner in CBSE, the developer searches the web for OTS communities and selection support tools. OTS-Wiki appears as a matching portal for most his searches. He decides to register at OTS-Wiki and start browsing the content for interesting information. In the beginning, the browsing is performed randomly to get an overview of the portal's functionality and information base. After a while, the developer tries to perform a few searches based on some requirements (wanted technologies and categories). The resulting list of components is large, providing information about a wide range of different components. As the familiarity with the portal and its features increases, the developer is capable of narrowing the set of potential components down to a feasible list. To make the final decision, he reads the

[1] The Cosiportal Project

reviews and evaluations performed by other users of the components. He also posts questions regarding the components in the related forums and chat rooms. Suddenly, another devloper with many years of OTS experience answers one of his messages, suggesting a strategy suitable for a start-up OTS developer. He suggests using components with high ratings and positive evaluations in the areas of installation and documentation, as they are quick to get up and running and provide good support. Using the OTS-Wiki evaluations, the developer ranks his findings based on the criterias suggested. The two top ranked components are then selected for further study and trial.

The beginner OTS developer has found enough information, both on his own, and by talking to other developers, to begin testing a few relevant components. A while later, the developer posts his own experiences in OTS-Wiki to help others in his situation.

2.2.2 EurOTS

This scenario takes advantage of the OTS-Wiki infrastructure, developing a feasible and incremental European Software Components Web-encyclopedia to foster the reuse of OTS components produced by European Community funded projects. Such projects often produce many software components suitable for reuse, including middleware, frameworks, libraries, testing software, etc. CORDIS offers a powerful knowledge tool to report the generated results from European funds[2]. OTS-Wiki can complement this technology in order to discover, search and select new software pieces from the whole of these projects. Moreover, re-users and citizens will have a powerful instrument to explain their needs and opinions about software. The specific objectives of the EurOTS scenario are:

1. Obtain in an incremental and collaborative way a knowledge base of the software components produced by projects funded by the European Community (NOT a software repository, but a meta-data repository).
2. Provide users with intelligent methods to help in the hard task of searching, comparing, evaluating and selecting OTS components developed in these projects.
3. Provide the European Council with new approaches to evaluate different dimensions as the reliability, reusability, etc. of the OTS produced by projects funded by them.
4. Use the developed OTS-Wiki as a collaborative tool to incrementally obtain a general knowledge base of non-European projects OTS from which the European society can benefit.

2.2.3 IT consultants for SMEs

A community of IT consultants offering services in COTS selection has evolved in Luxembourg and Belgium since 2001 (part of the CASSIS Network). The CASSIS Network[3] is aimed at developing a market for quality services in IT consultancy and promoting the quality of information systems in companies, particularly in Small and Medium Enterprises (SME) and public departments. In this context, the needs for knowledge sharing and knowledge reuse of OTS solutions are identified. The principles of OTS-Wiki can be introduced in this community to demonstrate knowledge sharing in selection of OTS software solutions. The specific objectives are:

[2] http://istresults.cordis.europa.eu/index.cfm?section=home&tpl=home
[3] http://www.cassis.lu/

1. Identify consultants in the CASSIS Network who demonstrate an interest in using the OTS-Wiki portal.
2. Provide training sessions to the selected consultants to help them understand the concepts, processes, and rules for using OTS-Wiki.
3. Give early access to the tool to these consultants.
4. Provide support for using the tool during their mission (mainly authoring and consumption of knowledge).
5. Analyze the impacts of the use of OTS-Wiki compared to other projects that do not use the OTS-Wiki.

This scenario can also be extended outside Belgium and Luxembourg, since methods of the CASSIS network are being promoted in other European countries[4].

[4] www.eu-certificates.org

Part II

Prestudy

Chapter 3

Theoretical Background and State-of-the-Art

In this chapter, we will present the theoretical background of our master project. A State-Of-The-Art (SOTA) is established to describe the terms and technologies relevant to our work. In addition, SOTA can highlight relevant issues and challenges motivating the research and development.

3.1 Component-Based Software Engineering

Component-based Software Engineering (CBSE) is a software engineering strategy where systems are developed using reusable parts (components). This includes developing components, and maintaining and improving component-based systems (CBS) by means of component replacement and customization [CBSE, 2006]. A software component is an encapsulated piece of software with well-defined interfaces enabling cross-component communication [Voas, 1998, Dean and Gravel, 2002]. The well-defined interfaces makes it possible to develop the system using components acquired from different sources. This is known as *Off-the-Shelf* (OTS) components. An OTS is defined as: "*a software product that is publicly available at some cost or with some licensing obligations, and other software projects can reuse and integrate it into their own products*" [Torchiano and Morisio, 2004].

Systematic reuse of OTS components may increase productivity and shorten time-to-market (TTM) compared to traditional software engineering strategies [Li et al., 2004]. Reuse is a recognized way to disseminate best-practice standards and expertise.

In this section we characterize both Commercial Off-The-Shelf (COTS) and Open Source Software (OSS) components, and look at risk management in CBSE.

3.1.1 COTS Components

COTS components are software components that are developed by commercial vendors and made available for lease or sale to the general public[1] [Morisio and Torchiano, 2002]. Using

[1] http://en.wikipedia.org/wiki/Commercial_off-the-shelf

COTS may be both time-saving and enhancing to a company's systems. The COTS vendors are experts in the area of functionality they offer, and may provide higher quality for a lower price than in-house development of the equivalent components [Voas, 1998]. Another benefit of COTS components is that commercial vendors often offer specific support programs and differential price structures suitable for each customer.

While COTS often provide efficient solutions to generic problems, especially in large corporate and government systems, the use of COTS may be problematic in some cases. The components are made generally to provide functionality to a wide range of potential customers. Often, different customers have slightly different requirements to the same components. The time saved on developing the component's functionality may then be consumed by integration and adaptation tasks. Also, the COTS vendor are responsible for the maintenance and evolution of their components, leaving the control of pieces of a company's software in the hands of third-party actors.

3.1.2 OSS Components

OSS components are software components developed by commercial vendors or open source communities. The difference between COTS and OSS components is that OSS components are released under some sort of open source license. Thus, the source code is available for anyone to use or change [Øyvind Hauge and Røsdal, 2006]. The main advantages of OSS components are:

- Free to acquire and use
- Available source code makes it possible to change the components
- Less vendor support risks

The main disadvantages of OSS components are:

- Changes are often needed as the components rarely come "good to go"
- Documentation and support may be lacking

More on open source in Section 3.4.

3.1.3 Risk Management in CBSE

Using OTS components introduces additional risks to a software development project. As more and more IT companies start to use OTS components, new strategies are required to manage the additional risks. Table 3.1 describes some possible risks in CBSE as presented by [Li et al., 2005]'s survey on risk management in OTS component-based development.

Table 3.1: Risks in CBSE

ID	Possible risks
R1	The project is delivered long after schedule
R2	Effort to select OTS components is not satisfactory estimated
R3	Effort to integrate OTS components is not satisfactory estimated
R4	Requirements are changed a lot
R5	OTS components are not sufficiently adapted to changing requirements
R6	It is not possible to (re) negotiate requirements with the customer, if OTS components are not satisfying all requirements
R7	OTS components negatively affect system reliability
R8	OTS components negatively affect system security
R9	OTS components negatively affect system performance
R10	OTS components are not satisfactory compatible with the production environment when the system is deployed
R11	It is difficult to identify whether defects are inside or outside the OTS components
R12	It is difficult to plan system maintenance, e.g., because different OTS components has asynchronous release cycles
R13	It is difficult to update the system with the latest OTS component version
R14	Provider do not provide enough technical support/training
R15	Information on the reputation and technical support ability of the provider are inadequate

Further, [Li et al., 2005] summarize a set of typical risk management strategies to manage the risks identified. Table 3.2 shows the strategies.

Table 3.2: Risk management strategies in CBSE

ID	Strategy
M1	Customer is actively involved in the "acquire" vs. "build" decision of OTS components
M2	Customer is actively involved in OTS component selection
M3	OTS components should be selected mainly based on architecture and standards compliance, instead of expected functionality
M4	OTS components qualities (reliability, security etc.) are seriously considered in the selection process
M5	Effort in learning OTS component is seriously considered in effort estimation
M6	Effort in black-box testing of OTS components is seriously considered in effort estimation
M7	Unfamiliar OTS components are integrated first
M8	Do integration testing incrementally (after each OTS component was integrated)
M9	Local OTS-experts actively follows updates of OTS components and possible consequences
M10	Maintain a continual watch on the market and look for possible substitute components
M11	Maintain a continual watch on provider support ability and reputation

[Li et al., 2005] concluded that the risk management survey could be summarized into three categories:

- Risk related to cost-estimation and changes in requirements happened more frequent than quality risks regarding the OTS components.
- The most used risk management strategies used were those related to incremental testing and trict OTS component quality evaluation.
- If the integrator seriously considered the possible effort on the quality evaluation of OTS components, it helped to solve the effort estimation risks in the OTS selection and integration.

Careful planning and resource-allocation are key requirements for successfully selecting and using OTS components. This is backed up by a study of COTS-based software development in the Norwegian IT industy in 2004 [Li et al., 2004]. The most important results were:

- Use of COTS can be done without changing or adapting existing software development processes.

- New activities, such as the *build vs buy decision*, the *COTS component selection*, and the *COTS component integration* are required to ensure successful use of COTS components.
- A new role, the *knowledge keeper*, storing knowledge and lessons learned from previous COTS project, may considerably increase the efficiency and success of using COTS components.

3.1.4 Summary

Developers that use OTS components in their work is the main potential user group of OTS-Wiki. It is the real users of the components that has the "hands-on" experience and information needed to evaluate the components. Thus, the quality of the content is dependent of the involvement from CBSE developers, communities, and businesses. The risks described above form some of the motivations for developing OTS-Wiki, providing a easy and predictable integration of OTS components.

3.2 GOThIC

GOThIC (Goal- Oriented Taxonomy and reuse Infrastructure Construction) is a goal oriented method for building and maintaining an infrastructure of Commercial-Off-The-Shelf (COTS) components by organizing market segments as a taxonomy. [Ayala and Franch, 2006]

Figure 3.1 shows the conceptual model of how the knowledge is organized based on the GOThIC method. The taxonomy is composed of two different types of nodes in this model. This is market segments and categories. The market segments groups the basic types of COTS into different domains, e.g., such as anti-virus tools and spreadsheet applications. The domains covers a pretty large group of functionality to prevent too many domain groups with a limited coverage area. A component may also cover more than one market segment. The category groups related market segments and subcategories.

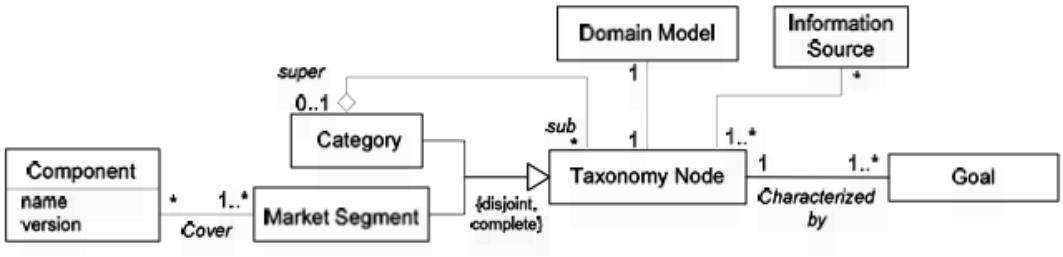

Figure 3.1: Conceptual model of the GOThIC method

The GOThIC method has been structured into seven activities:

Exploration of information sources
This activity consists of gathering relevant information, including type, supporting media, cost, etc. *Gathering of sources*, *Analysis of sources* and *Prioritisation of sources* are the three related sub-activities.

COTS marketplace domain analysis
The basic elements of the domain are identified during the domain analysis, and an

understanding of the relationships among these elements are organized and represented. The lack of a standard terminology in COTS frameworks leads to confusion as similar components are denoted differently by different vendors. The domain analysis should help preventing these differences.

Four artifacts for recording and representing the knowledge from domain analysis are proposed in GOThIC. These are *Use Case Specification*, *Class Diagram*, *Quality Model*, and a *Glossary of terms*. Figure 3.2 illustrates these artifacts.

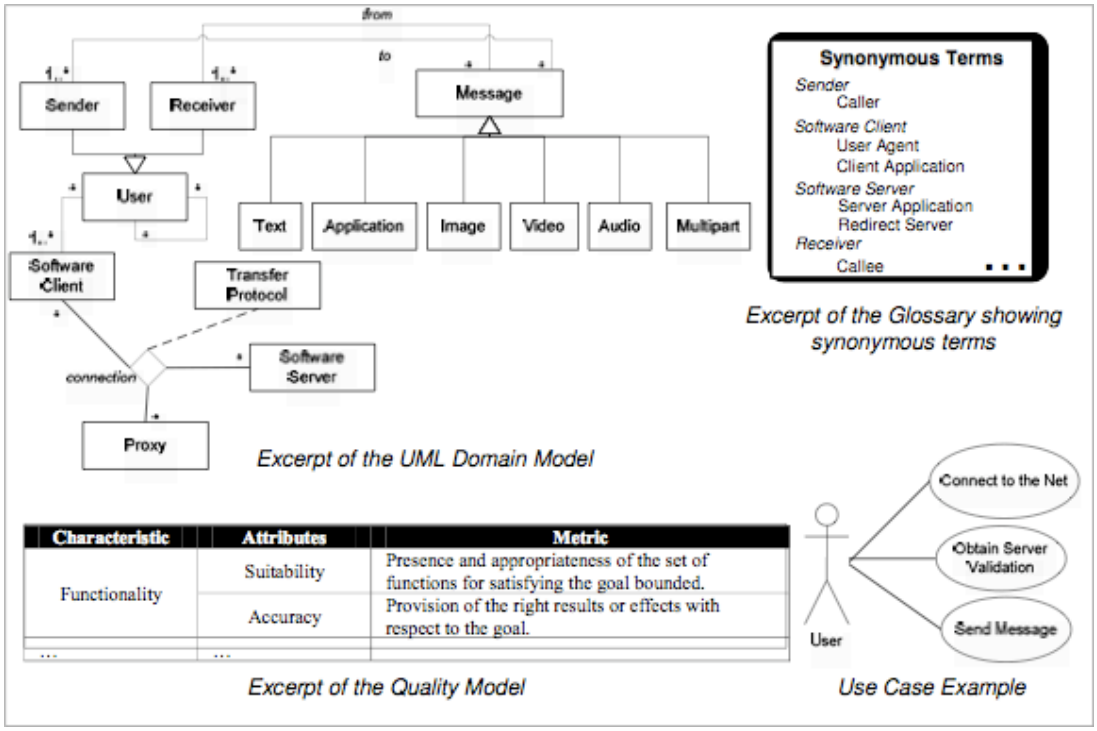

Figure 3.2: Excerpts of the four types of artifacts

Identification, refinement, and statement of goals

The activities in this stage are *Identification*, *Refinement* and *Statement*. These are performed iteratively. The goal or objective is formulated through these activities. Table 3.3 [Ayala et al., 2007] shows an example of a goal statement.

Establishment of dependencies

A COTS component may need other components for several reasons. This may be to enable its functionality, to complement its functionality, or to enhance its quality attributes. The dependencies are identified from analyzing the goal information from the previously performed activities.

Goal taxonomy structuring

The taxonomies in GOThIC are goal-driven. The goals are operationalized by variables such as number of users and data processing profile. In this case, the variables represent classifiers with values. A subcategory or marked segment applies to each possible value these classifiers may have.

Table 3.3: An example of goal statement

Goal:	**Multiuser Textual Communication Established**
Type	Achievement
Description	Provide RTSC in a Text Multi-user Environment
Agent	Software Client
Stakeholder(s)	Software Client, Software Server, Sender, Receiver
Precondition(s)	1. Users Communicated in Real Time; 2. Session Established; 3. Number of users >=2
Postcondition(s)	Multiuser Textual Communication Established
Subgoal(s)	1. Software Client Provided; 2. Software Server Provided

Taxonomy validation
The taxonomy needs to be consistent, complete and not ambiguous to be useful in the search process. This activity describes four steps to ensure these conditions.

Knowledge base management
The GOThIC method provides mechanisms for maintaining a knowledge repository through the definition found in the UML class diagram. The infrastructure from this knowledge base helps to ease the evolution and maintenance of the taxonomies.

3.2.1 Summary

Table 3.4 [Ayala et al., 2007] summarize many of the challenges of the OTS selection process. By using the GOThIC method in OTS-Wiki, we are able to solve many problems related to these challenges. The GOThIC method provides taxonomies to organize all information related to a OTS component, and to reuse this information structure. [Ayala et al., 2007]

There are heavy start-up costs related to the GOThIC method. The article [Ayala et al., 2007] proposes to combine the GOThIC method with the productive potential of open source collaboration principle. The OTS users can then work as a community building and maintaining a OTS information repository, which is one of the visions of OTS-Wiki.

3.3 DesCOTS

DesCOTS (Description, evaluation, and selection of commercial-off the shelf (COTS) components) is a software system with several tools interacting to provide support in the different activities in the process of selecting COTS software components. This includes both functional and non-functional aspects of the system. [Grau et al., 2004]

Table 3.4: Role-related challenges supporting OTS selection

OTS User Role	Current Practice	Problem	Challenge
Market Watcher (MW)	• Proliferation of cataloguing initiatives from profit and non-profit organizations. • Catalogues contain only brief and unstructured descriptions of some inventoried components. • Components characterization usually presents a hierarchy of items without a clear rationale behind.	• Understanding and use of the categorization proposals may be difficult. • Several descriptions of the same component.	• Understandable Taxonomies [18] • Common Component Description Metamodel [6]
Quality Engineer (QE)	• COTS vendors and specially OSS community projects do not provide structured and enough information for supporting evaluation and quality assessment of their products.	• Structuring and discovering important information leads to rework, confusion, missing critical information (as interoperability) and possibly deterring the use of some components.	• Component Description Metamodel embracing quality characteristics [17]
Selector (S)	• Non-Technical information about the components is even more difficult to be located.	• Hard Requirements negotiation. • Hard identification of mismatches among components characteristics and the requirements in order to support the final decision	• Component Description Metamodel embracing non-technical factors [19]
Knowledge Keeper (KK)	• No support for organizations (mainly small and medium) that needs to carry out continuously OTS selection processes to reuse their knowledge about them.	• Reuse of knowledge is usually tacit, leading to be lost if people are replaced.	• Reuse Infrastructure Support [20]

The selection process builds upon the principle of comparing user requirements with evaluations of COTS components, with a focus on quality requirements. Quality Models are then constructed of several quality factors. These factors may differ significantly between the different COTS domains, and therefore it is important to find the quality factors that are best suited for the specific description task. [Grau et al., 2004]

3.3.1 The DesCOTS Tools

As introduced above, the DesCOTS system consists of several different tool to support the OTS software component selection process. Below follows a short description of these.

Quality Model Tool

The Quality Model Tool provides functionality to define software quality factors to reuse these in different models, to state relationship among the quality factors, to assign metrics for their future evaluations, and to define Requirement Patterns to ease the final stage of the selection process.

It is often a time demanding task to construct a Quality Model. The Quality Model is not made from scratch, but is built upon the quality framework provided by the ISO/IEC 9126-1 quality standard.

COTS Evaluation Tool

The Evaluation Tool uses the Quality Model from the suited domain to evaluate the COTS component candidates.

COTS Selection Tool

The Selection Tool provides support for two different processes. It first supports the process of defining the selection requirements. Then it analyses these requirements and the evaluations of the COTS components to assist the selection of a component.

The selection requirements are stated according to the Quality Model for the specific COTS domain, either from the defined Requirement Patters or as new ones. When the requirements list is ready, the selection process tool provides a result of possible components matching these criteria.

Taxonomy Tool
Many quality factors may be used in the different COTS domains. The Taxonomy Tool is integrated in the Quality Model Tool to provide support for reuse of Quality Models. When introducing a new COTS domain to the tool, it is first placed at the right place in the taxonomy. The tool then starts to create the Quality Model by adopting all quality factors belonging to the category. The user may now add additional quality factors belonging to that specific domain.

Other tools related to software component selection:

- MiniSQUID
- OPAL
- eCOTS
- IRqA

3.3.2 Quality Models - Conceptual model

The conceptual model of Quality Models is here divided into the nine following parts [Franch et al., 2007]:

Arranging quality models in a taxonomy of categories and domains
The quality models are organized by using a taxonomy which is divided into domains and categories. The domains are ordered into categories, and the categories are grouped into other categories. Examples of categories is communication infrastructure and collaboration software. Domains may be workflow systems, mail servers, anti-virus, etc.

Classifying general objects
By introducing a class named Object, there are kept some common attributes between the different elements. The general objects are Quality Pieces, Quality Entities, Metrics, Quality Entity Metrics Assignments, Requirement Patterns, Families of Patterns, and Relation Scales.

Defining and grouping of quality entities
Quality models contain quality entities to cope with a specific quality scope.

Quality entities are grouped into quality fragments.

Quality patterns are a collection of quality entities that appears in many quality models. These patters are placed in a pattern catalog for reuse.

Defining hierarchies of quality entities
ISO/IEC 9126-1 is chosen as the standard for the quality framework. This standard uses a hierarchical structure. The DesCOTS conceptual model does not restrict number of levels in the hierarchy, and attributes or subcharacteristics may be associated to several subcharacteristics.

Defining metrics
There are two types of metrics, global metrics and quality model specific metrics. The metrics are classified to qualitative metrics and quantitative metrics. It is qualitative when it is not possible to get a precise value but only a subjective estimate, and quantitative if the measure is a specific value.

Assignment of metrics to quality entities
Metrics must be assigned to the quality entities so that the evaluation of a component can follow a quality model.

Establishing relationships among quality entities
There should be established relationships among quality entities of a quality model version to avoid conflicts between quality attributes.

Defining requirement patterns
A requirement pattern is used as a template to support the creation of software requirements. These patterns are put into so called families of patterns, and may also organized in a hierarchical structure of many pattern families.

Defining a glossary
The glossary consists of terms used when constructing quality models. This glossary is global.

3.3.3 Summary

The article [Ayala et al., 2007] suggests integrating the DesCOTS system into OTS-Wiki. DesCOTS includes a set of tools that interoperates to support the OTS selection process: the Quality Model Tool, the COTS Evaluation Tool, the COTS Selection Tool, and the Taxonomy Tool. By integrating the DesCOTS system in OTS-Wiki, we could take advantage of these tools and provide systematic support the whole OTS selection process. See Figure 5.1 in Chapter 5 for OTS-Wiki interaction with the DesCOTS system.

3.4 Open Source

In the early age of the computers, most software were developed by scientists and engineers. Sharing the code as well as research results was a normal practise among the these persons [von Hippel and von Krogh, 2003]. The sharing of code allowed others to change and improve code written by other developers. This code sharing practise from the early computer days makes up the foundation of the open source community we know today.

Open source is defined by Wikipedia[2] as the principles to provide open access to the production and design process. The open source term is normally applied to the source code of computer software. The Open Source Initiative (OSI)[3] is a non-profit organization managing the Open Source Definition (OSD)[4]. According to OSD, the distribution terms must follow ten criteria, in addition to free access to the source code. The definition origins from "The Debian Free Software Guidelines" written by Bruce Perens. The Debian spesific content was removed to create the OSD. The definition is currently at version 1.9.

[2] http://en.wikipedia.org/wiki/Open_source
[3] http://www.opensource.org
[4] http://www.opensource.org/docs/definition.php

The Free Software Foundation (FSF)[5] defines four kinds of freedom in the Free Software Definition:

- The freedom to run the program, for any purpose (freedom 0).
- The freedom to study how the program works, and adapt it to your needs (freedom 1). Access to the source code is a precondition for this.
- The freedom to redistribute copies so you can help your neighbour (freedom 2).
- The freedom to improve the program, and release your improvements to the public, so that the whole community benefits (freedom 3). Access to the source code is a precondition for this.

3.4.1 Examples of Open Source Projects

Many well know software products are released under open source licenses. Linus Thorvalds started working on the **Linux** operating system in 1991, a Unix-like operating system for PCs. Later the same year, he released the source code of the Linux kernel into a use-group on the Internet [Weber, 2004]. He encouraged people to help by commiting modules and comments to the kernel. This attracted a lot of attention, and led to the release of Linux 1.0 in 1994. The model which Linux was developed by was at this time unique, with a large numbers of volunteers contributing to the project through the Internet[Raymond, 2001]. The Linux project is today one of the most successful open source projects, and the the popularity is steadily increasing. The Linux kernel is licensed under GPL.

Another successful open source project is the **Apache** project. Apache is a web server application with a market share of 56% [Netcraft, 2007]. Rob McCool left the development team of the National Centre for Supercomputing Applications (NCSA) http server in 1998, and the development of the server stopped. Many web masters had made their own patches and contributions to the server. Some of these people later formed the Apache Group to coordinate the distribution of the patches [Fielding, 1999]. All modifications made to the server were controlled by the Apache Group through a voting system. The Apache Group were later named Apache Software Foundation (ASF)[6]. This foundation is a non-profit organization with several open source projects such as the Apache server, ANT, Tomcat and many others. [Øyvind Hauge and Røsdal, 2006]

Netscape released the source code of their browser in 1998. The project was named **Mozilla**. The code base for the Mozilla project was originally released under the Netscape Public license. This license was updated and renamed Mozilla Public license[7].

3.4.2 Open Source Licensing

Open source software uses a lot of different license types. These license types are to a large extent based on GNU General Public License, the GNU Lesser General Public License, and the Berkley Software Distribution License(s) .

[5] http://www.fsf.org/
[6] http://www.apache.org
[7] http://en.wikipedia.org/wiki/Mozilla

GNU General Public License (GPL v.2)

The GPL[8] [9] license is the most commonly used open source license today, and was introduced by Richard Stallman in 1989 for the Free Software Foundation (FSF). The license intends to guarantee the freedom to share and change the software code. This freedom is provided by the following rights:

- The right to freely distribute copies, both as-is and in modified versions
- The right to modify the source code
- The right to run the program for any intended purpose

The GLP license imposes that all software using GPL licensed source code, must also be licensed under GPL. Because of this restriction, it is almost impossible to incorporate GLP licensed source code into proprietary closed source software. The license does allow anyone to charge a fee for distributing GPL software.

GNU Lesser General Public License (LGPL v.2.1)

LGPL[10] [11] was introduced together with GPL v.2. LGPL is less restrictive than the GPL license as it permits the linking of GPL licensed libraries into non-free programs. Section five in the LGPL license states the following concerning this:

> A program that contains no derivative of any portion of the Library, but is designed to work with the Library by being compiled or linked with it, is called a "work that uses the Library". Such a work, in isolation, is not a derivative work of the Library, and therefore falls outside the scope of this License.

LGPL was introduced to give a larger group of users access to the GNU libraries.

Berkley Software Distribution License (BSD)

The BDS license was originally formulated at the University of California Berkley to use with the Berkley Software Distribution, a Unix-like operating system. The first version was revised, and the result of this revision is called a modified BSD licenses.

This BSD license[12] has few restrictions compared to other free software licenses such as GPL and LGPL. The BSD license allows proprietary commercial use, and there are for example BSD networking code in Microsoft products, and FreeBSD[13] components in Mac OSX.

The BSD license is sometimes referred to as *copycenter*, as in:"Take it down to the copy center and make as many copies as you want." This is in comparison to standard copyright and copyleft[14].

[8] http://www.gnu.org/licenses/gpl.html
[9] http://www.opensource.org/ licenses/gpl-license.php
[10] http://www.gnu.org/licenses/lgpl.html
[11] http://www.opensource.org/licenses/lgpl-license.php
[12] http://www.wikipedia.org/BSD_licenses
[13] http://www.freebsd.org/
[14] Wikipedia defintion: Copyleft is a play on the word copyright and is the practice of using copyright law to remove restrictions on distributing copies and modified versions of a work for others and requiring that the same freedoms be preserved in modified versions. (http://en.wikipedia.org/wiki/Copyleft)

3.4.3 The Apache License

The Apache License is similar to the BSD license, and it allows the code to be copied, modified and redistributed with a few restrictions. This includes redistribution under other license types, as long as the restrictions stated in the Apache License is followed
[Øyvind Hauge and Røsdal, 2006]. The Free Software Foundation considers the Apache License to be incompatible with GPL[15].

3.4.4 Mozilla Public Licence

Mozilla Public License (MPL) was created together with the release of the Netscape source code. Unlike GPL, MPL allows code licensed under MPL to be released in combination with code with other licenses [Øyvind Hauge and Røsdal, 2006]. Code copied or changed under MPL must stay under the MPL. Some of the restrictions on MPL however, makes it incompatible with GPL[16].

3.5 Social Computing

Recently, we are beginning to see vast and sudden changes in the way people are using the Web. The evolution of Web 2.0 has introduced new and exciting opportunities for rich web applications [Hinchcliffe, 2006]. One important usage we see is *Social Computing* [Charron et al., 2006]. Microsoft's Dr. Gary Flake referrs to this as *Internet Singularity*:

> *"The idea that a deeper and tighter coupling between the online and offline worlds will accelerate science, business, society, and self-actualization."*

The evolution towards Internet Singularity is depicted in Figure 3.3.

IBM Social Computing Group [IBM, 2007] characterizes social computing thus:

> *"The central hallmark of social computing is that it relies on the notion of social identity: that is, it is not just the data that matters, but who that data 'belongs to', and how the identity of the 'owner' of that data is related to other identities in the system. More generally, social computing systems are likely to contain components that support and represent social constructs such as identity, reputation, trust, accountability, presence, social roles, and ownership."*

Popular web sites, such as YouTube[17], MySpace[18], and Facebook[19] have become vastly popular and have shifted the way we interact socially on the web [Geelan, 2006]. Especially younger people spend a significant part of their time using such social software. New web communities and portals need to learn from this trend when hoping to gain popularity and a growing base of re-users. Thus, OTS-Wiki will benefit from both Web 2.0 technologies, and social computing principles.

[15] http://en.wikipedia.org/wiki/Apache_License
[16] http://en.wikipedia.org/wiki/Mozilla_Public_License
[17] http://www.youtube.com/
[18] http://www.myspace.com/
[19] http://www.facebook.com/

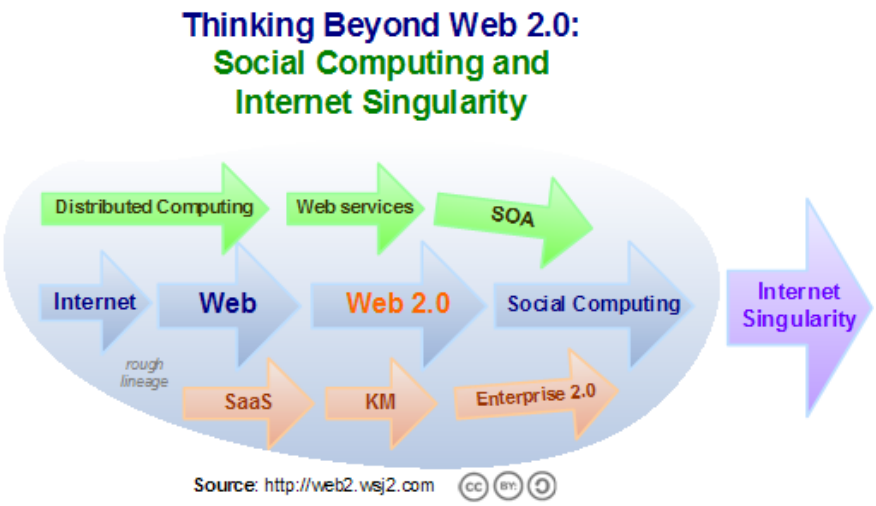

Figure 3.3: Towards Internet Singularity

3.5.1 Wiki

A wiki (from the Hawaiian Wikiwiki meaning “fast") is a web-site that allows visitors to add, remove, edit, and change content, typically without the need for registration. It also allows for linking among any number of pages[20]. This simplifies interaction and operation of such sites and facilitates mass-collaboration on the web. With open collaboration across geographical locations becoming more and more important, the wiki solution has grown in popularity.

The WikiWikiWeb[21], introduced in 1994, was the first site to adopt the wiki principle. Since then, the wiki engine has acted as a foundation for numerous web-sites and web communities. Wikipedia, the free encyclopedia[22] is perhaps the most well-known and widely used wiki solution. Wikipedia has more than 7 million articles in over 250 languages worldwide (April, 2007). All the articles are written collaboratively by volunteers from all over the world.

3.6 Software Development Processes

3.6.1 Scrum

Scrum is an agile method for managing the system development process, with focus on being flexible, adaptable and productive. The term “scrum” originates from a term in the sport of rugby, where it means to get a ball which is out of play into the game again.

As a part of Scrum’s agile nature, it does not require specific engineering practices. It can adopt to the development techniques already used in the organization. The Scrum idea builds upon that the system is based on some environmental and technical variables (requirements, resources, available time etc.). These variables are changing during project life-cycle, which makes the development process complex and unpredictable. Scrum aims to improve the prac-

[20] http://en.wikipedia.org/wiki/Wiki
[21] http://c2.com/cgi/wiki
[22] http://www.wikipedia.org/

tices by involving frequent managing activities. Scrum is best suitable for small project teams consisting of less than 10 members. If more peoples are available, they should be divided into multiple teams. Figure 3.4 illustrates the activities involved in Scrum-based software development [Abrahamsson et al., 2002].

Processes

In Scrum there are three phases. These are as follows [Abrahamsson et al., 2002]:

The pre-game phase includes the two sub-phases *planning* and *architecture/high level design*.

A Product Backlog list is made in the planning phase. The Backlog contains all currently known requirements. These are then prioritized and an estimate of the upcoming workload is created. During the development the list is updated constantly. The Scrum Team reviews the list after each iteration. Setting up a project team, resource needs, and required training are also important activities in the planning phase.

The high level design based on the Product Backlog is set up during the architecture phase. The design is reviewed in a meeting, and decisions are made, based on this. A preliminary plan for the releases are also made.

The development phase or the game phase is treated as "black box", meaning the unpredictable is expected. During this phase the system is developed in Sprints. Sprints are an important concept in the Scrum method. A Sprint is the procedure of accommodating to the changing environmental variables such as time frame, quality, requirements, resources and implementation technology and tools. It lasts approximately one month. Traditional phases such as requirements, analysis, design, evolution, and delivery are also included in Sprint. The duration is between one week and a month.

The project team uses tools such as Sprint Planning Meetings, Sprint Backlog and Daily Scrum meetings. On the last day of the Sprint, a Sprint Review meeting is held, and the result is presented to the persons involved.

The post-game phase is the end of the release, and is entered when it is agreed upon that the requirements is fulfilled. Integration, testing and documentation are done in this phase.

Roles

The Scrum Master is responsible for following the practices and rules of Scrum during the project, a sort of manager role. The **Product Owner** is selected by the Scrum Master, the customer and the management, and is responsible for the project. He also makes the final decisions when it comes to the Product Backlog list. **The Scrum Team** is the project team, and is responsible for each Sprint. **The customer** takes part in things related to the Product Backlog list. **Management** has the last word when it comes to decision making, and also participates in the requirements process.

3.6.2 eXtreme Programming (XP)

Extreme Programming (XP) [Beck, 1999, Abrahamsson et al., 2002] is a lightweight, test driven and low-risk way to develop software. XP emphasizes the simplest possible solutions. It is a collection of practices taken from other methodologies. The business decisions are taken by the customer, while the technical issues are in the hands of the programmers. XP fits when the development team is small or medium sized, meaning a maximum of twenty project team

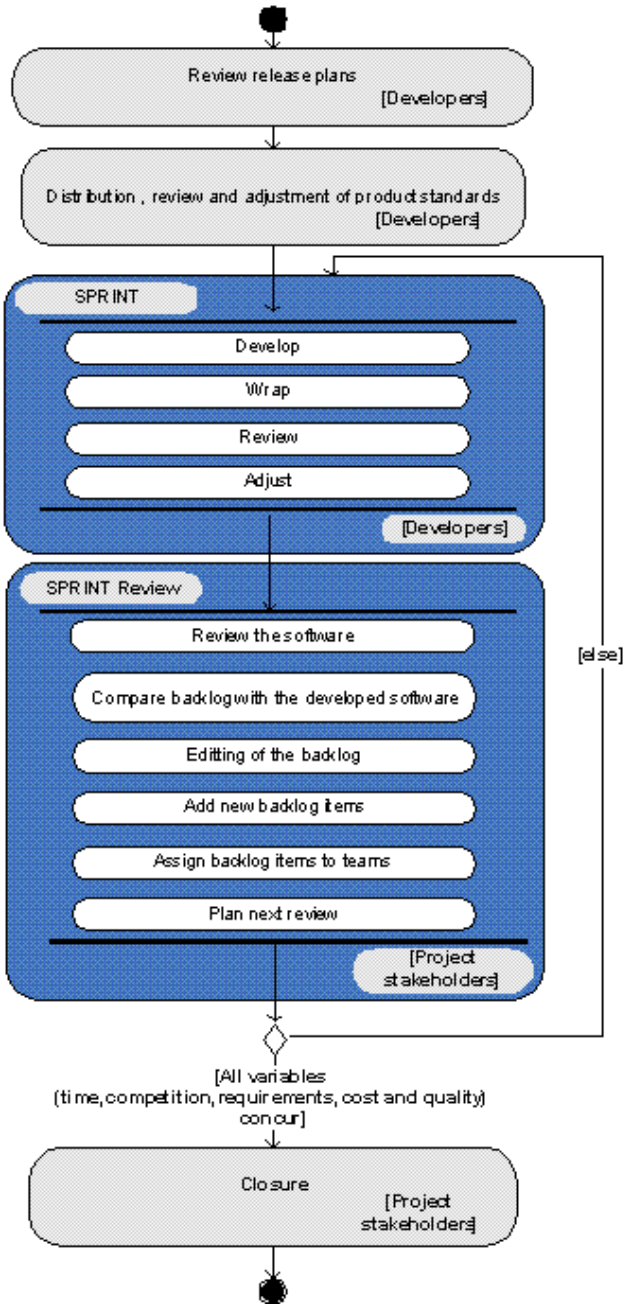

Figure 3.4: The Scrum metamodel

members. The short iterations make it ideal when the requirements are changed rapidly during development. The interaction between the customer and the programmers is also an important aspect of XP.

Figure 3.5 [Sørensen, 2002] illustrates the phases of XP.

Other features known from XP:

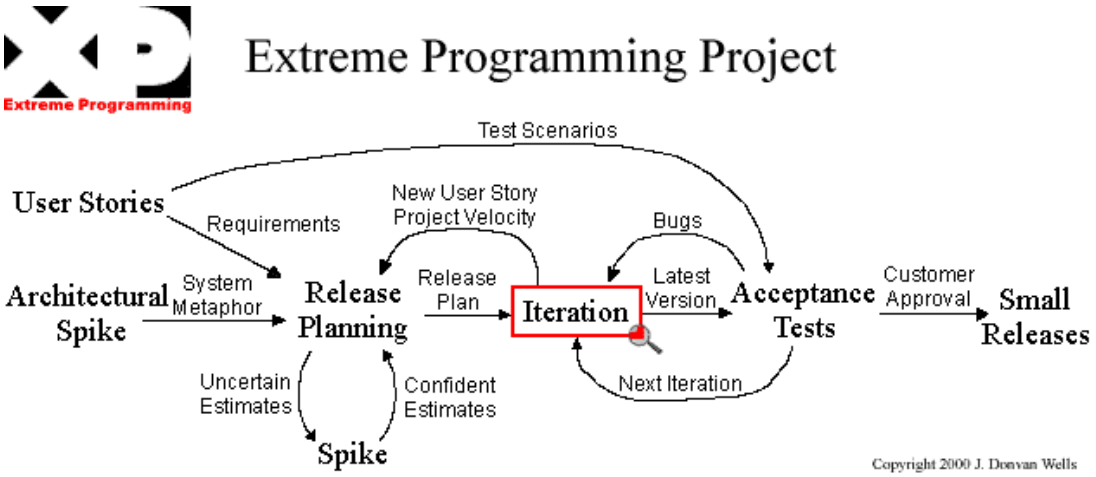

Figure 3.5: Extreme Programming project

- Pair programming is important in XP, meaning two persons sitting together in front of one computer when programming.
- Collective ownership so that anyone can make changes to anything in the code.
- Maximum of 40-hour work per week, and no overtime two weeks in a row.
- Communication between the members should be possible all the time, and the customer should always be available to the team.
- The coding is based on rules and standards, so that communication through the code is made easier.

Processes

The XP life cycle consists of six phases.

Exploration The customer tells the programmers what they want to be made by writing stories. Each story tells a feature request that should be implemented in the system, In parallel to this, the project team uses the time to get familiar with the needed tools, technology and practices to be used in the project. Duration of this period is usually between a few weeks to a few months, dependent on how well the team members know the technology.

Planning In the planning phase the priority of the stories made in the exploration phase is set, and a delivery schedule is set up. The first release should be delivered between two and six month later. The phase takes a few days.

Iterations to release The schedule made in the planning phase is divided into several iterations with an implementation length between one to four weeks. It is up to the customer to choose which stories to be used for each iteration. When an iteration is ended, the customer should run functional tests to verify the result of the iteration.

Productionizing In this phase the software typically goes through more testing to certify that it is ready to be released to the customer, The iterations usually goes down to one week length. Performance tuning may also be done in this phase.

Maintenance After delivering the first release to the customer, the system needs to be kept running and simultaneously new iterations must be produced. The maintenance phase may also include the incorporation of new members to the development team.

Death When the customer does not have any more stories, the death phase is introduced to the project life cycle. The customer is satisfied with the product, also concerning performance and reliability. No more changes are made, so the documentation is now written. If the system cannot deliver what is needed, or the economic situation force it, the death phase may occur before the system is completed.

Roles

During development, different roles apply to the XP method. The **programmers** write the tests and make the code as simple as possible. The **customers** write the stories and prioritize them, write the functional tests, and tell when the system is satisfactory. **Testers** help the customer write the tests, and they run them. The feedback in XP is given by the **tracker**, e.g., on whether the goals are reachable. The one responsible for the process is the **coach**, and he or she guides the other team members. The **consultant** is an external resource with specific field knowledge to guide the team to resolve certain issues. **Manager** or Big Boss makes the decisions in the project.

3.6.3 Summary

Agile software development methods are growing in popularity [Fraser et al., 2006]. Open source and agile methodology will be important guidelines in the development of OTS-Wiki. The software engineering part of this thesis will use elements from both Scrum and XP.

Chapter 4

Related Work

This chapter will describe some work and projects related to this thesis. Other web portals in the OTS community are important, but relevant student projects are also discussed.

4.1 Related Web Portals

There exists several web portals aiming to provide support in the process of selecting OTS components. Table 4.1 [Ayala et al., 2007] shows a overview of existing resources to support the OTS selection process. The abbreviations in the right column of the table means: Market Watcher (MW), Quality Engineer (QE), Selector (S), and Knowledge Keeper (KK).

The resources listed in this table covers different areas of the selection process. **Sourceforge.net**, **Tigris.org** and **Freshmeat** are well known web-portals providing functionality similar to the OTS-Wiki. **Ohloh** is another portal which gathers statistics related to the development of different open source software projects. These four web resources are presented in the following sections.

4.1.1 SourceForge.net

SourceForge.net[1] is web portal hosting open source software projects, where software developers can control and manage their projects free of charge. In January 2007, it had reached nearly 140000 hosted projects[2].

SourceForge.net offers a variety of services to manage and control a development project. This includes among others, source code version control systems like CVS and Subversion, file release systems, forums, mailing lists, publicity by news and statistics, and donation systems.

[1] http://www.sourceforge.net/
[2] http://en.wikipedia.org/wiki/SourceForge.net

Table 4.1: Web Resources Available for Supporting OTS Components Selection

Name	Characterization	Retrieval Schema	Scope	Component Information	Additional Information	Intended Objective	Support to the Roles			
							MW	QE	S	KK
COTS Vendors	-	-	COTS	Non-Structured (**NS**)	Requests (R)	Marketing	*	*	*	-
OSS Community Project	-	Browsing (B) Keyword search (KS)	OSS	**NS**	(R), Forums (F), Documentation (D)	Open Source Development	*	*	*	-
SourceForge.net	SC	B, KS	OSS	**NS**	Newsletter (N)	Promote and hosting OSS Community Projects	√	-	-	-
ComponentSource.com	-SC -Platforms	B, KS	Mainly COTS	**NS**	(N), (F), Demos	Marketing	√	-	-	-
Tigris.org	-SC	B	OSS-SE	**NS**	Messages from community owners, papers, etc.	Promote and hosting OSS for supporting Software Engineering (SE)	√	*	*	-
OpenCores.org	-SC	B, KS	OSS-IP	**NS**	(F), (N), Articles	Promote, design and hosting OSS for supporting IP-Core	√	*	-	-
KnowledgeStorm.com	-IT Solutions (ITS)	B, KS	Mainly COTS	**NS**	Case Studies Forecasts Vendor white papers, Demos.	Marketing	√	*	-	-
CMSmatrix.org	Name	List Selection	OTS-CMS	Semi-Structured (SS)	Comparison table	Open and free collaboration for indexing OTS-Content Management Systems (CMS)	*	*	-	-
Messangingmatrix.com	Name	List Selection	OTS-Messaging	SS	Comparison table	Open and free collaboration for indexing OTS-Messaging related systems	*	*	-	-
TheServerSide.com	Name	B, KS	Java	**NS**	Papers, News	Inform about trends in OTS-Java and promote marketing	√	-	-	-
Freshmeat.net	SC	B, KS	Mainly OSS	SS	Papers, Chat, (F)	Promote and hosting OSS Community Projects	*	*	-	-
Forrester.com	ITS	B, KS	Broad IT Solutions	**NS**	IT Support/ Not free	Selling IT Strategic Support	*	-	-	-
Gartner.com	ITS	B, KS	Broad IT Solutions	**NS**	IT Support/ Not free	Selling IT Strategic Support	*	-	-	-
Incose.org	SC	B, KS	COTS-SyE	SS	Research Trends	Research Support in Systems Engineering (SyE)	*	*	-	-

4.1.2 Tigris.org

Tigris.org[3] is a community hosting open source software projects. It offers services like web hosting, mailing lists, issue tracking, and revision control with Subversion. Tigris.org has it's main focus on projects for collaborative software development[4].

To get a project to be hosted by Tigris.org, it has to be approved by the administrators. It has to fit he Tigris.org mission, which is the development of software engineering tools.

Tigris.org also aims to provide a portal with no dead projects, through a commitment made by the developers to have a active developers cycle.

4.1.3 Freshmeat.net

Freshmeat[5] is a large collection of Unix and cross-platform software. The main focus in on open source software, but also closed-source and commercial software may be found on Freshmeat.

Developers register their project on Freshmeat, making it available for download. Users may give feedback to the developers by rating or commenting the projects[6].

[3]http://www.tigris.org/
[4]http://en.wikipedia.org/wiki/Tigris.org
[5]http://www.freshmeat.net/
[6]http://en.wikipedia.org/wiki/Freshmeat

4.1.4 Ohloh

Ohloh[7] Open Source Directory provides community driven content together with up-to-date information on open source projects generated by sources such as source code crawlers. These crawlers gathers project data based on the activity statistics in the revision control systems. Examples of such data is proportion of different programming languages in the source code, different licences, number of code lines in the code base.

Ohloh also provides a summary of this data, including project activity information and possible licence conflicts in the code base.

Users may give feedback to the project owners by reviews and ratings, as well as adding useful additional resources related to the different projects.

[7] http://www.ohloh.net/

Part III

The OTS-Wiki

Chapter 5

System Introduction

This chapter will introduce the OTS-Wiki and the contributions made in this thesis. We look at the starting point for the development, provided by [Ayala et al., 2007], and present the Java platform as an enabling technology for the implementation. We also present the Cosiportal Project, a related Master thesis by Per Kristian Schanke.

5.1 Contribution Overview

The main contribution of this thesis is the initial implementation of the OTS-Wiki portal. A specification of the requirements, both functional and non-functional, is needed to guide the implementation. The requirements specification (Chapter 6) will describe the functionality we want to implement in this thesis, emphasizing on the basic features needed to start using the portal. An overall architecture is presented in Chapter 7, including a data model to depict the data structure of the implemented portal.

In short, OTS-Wiki is a web portal aimed at fostering evaluation and selection of OTS components. The portal is enhanced by the wiki principle, allowing the community users to populate and edit the portal content. The implemented version of the portal will try to show the benefits of this enhancement and foster ideas and suggestions for further development of OTS-Wiki.

5.2 OTS-Wiki Starting Point

The initial work done by [Ayala et al., 2007] serves as a starting point for the work of this thesis. Figure 5.1 depicts the suggested main interactions in the portal. Here, any OTS Community user can use OTS-Wiki as a meta-portal for providing support to:

a) Locating OTS and information about them

b) Recording component information in a structured way

c) Maintaining and reusing such information

d) Getting tool support for performing selection processes

In addition, it integrates the DesCOTS system which includes a set of tools to support the whole OTS selection process. DesCOTS is described in detail in Section 3.3.

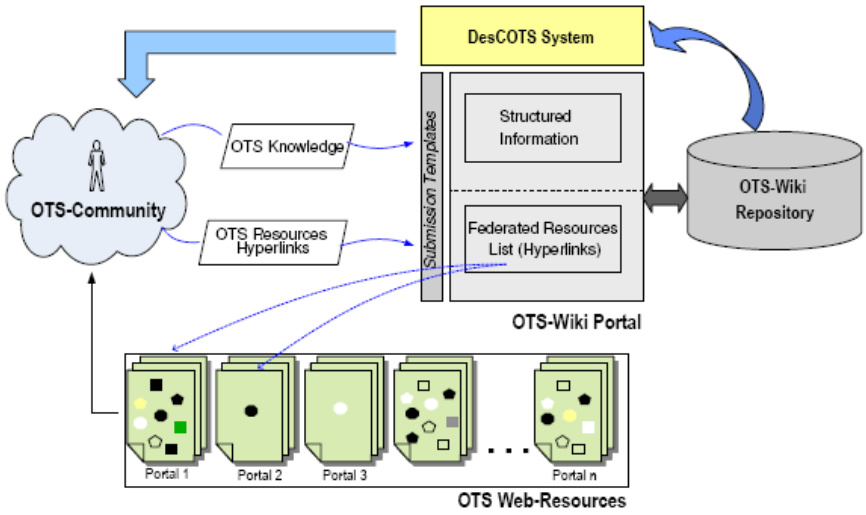

Figure 5.1: Suggested system interactions

5.2.1 Functionality Scenarios

[Ayala et al., 2007] define a set of scenarios to describe the wanted functionality in OTS-Wiki. The suggested scenarios are presented in Table 5.1. A simple prototype has been developed using a Wiki-based content management system from the Moddle community[1]. Figure 5.2 shows a screenshot of the prototype.

This thesis will use the suggested scenarios and prototype implementation as references in the further development of OTS-Wiki. We emphasize that our work also has elements of independent research, valuable to our field of study and interests, but that the principles of the work of [Ayala et al., 2007] are very important.

Table 5.1: Functionality scenarios

ID	Goal	Description
S1	Fostering OTS Community	OTS Technology users are encouraged to work as a high performance team for reusing and sharing OTS Components Information in an Open and Freely accessible OTS-Wiki Portal
S2	Incremental Population of Content	Users are encouraged to publish and share content they considered helpful to the OTS Community
S3	Federation of OTS Resources in OTS-Wiki	Users are encouraged to publish content that they consider may be helpful to the Community (Hyperlinks, references or files)
S4	Enabled Systematic Support for Selection Process	Tools are provided to support the OTS selection activities automatically, using the standardized data from the repository (DesCOTS functionality)
S5	Enabled Active Communication	Users are encouraged to maintain an active and fruitful communication among them using discussion boards and chat rooms
S6	Enabled Assisted Search	Users are provided with searching facilities to locate OTS components information using keywords or taxonomy navigation
S7	New Functionality Requested to the Community	Users are provided with a Requesting Board area for requesting information of components functionality that do not already exist in OTS-Wiki (but maybe in other portals) or new component functionality to the Community
S8	Enabled a Glossary Construction	Users are encouraged to detail the meaning of unknown or confusing terms

[1] http://moddle.org/

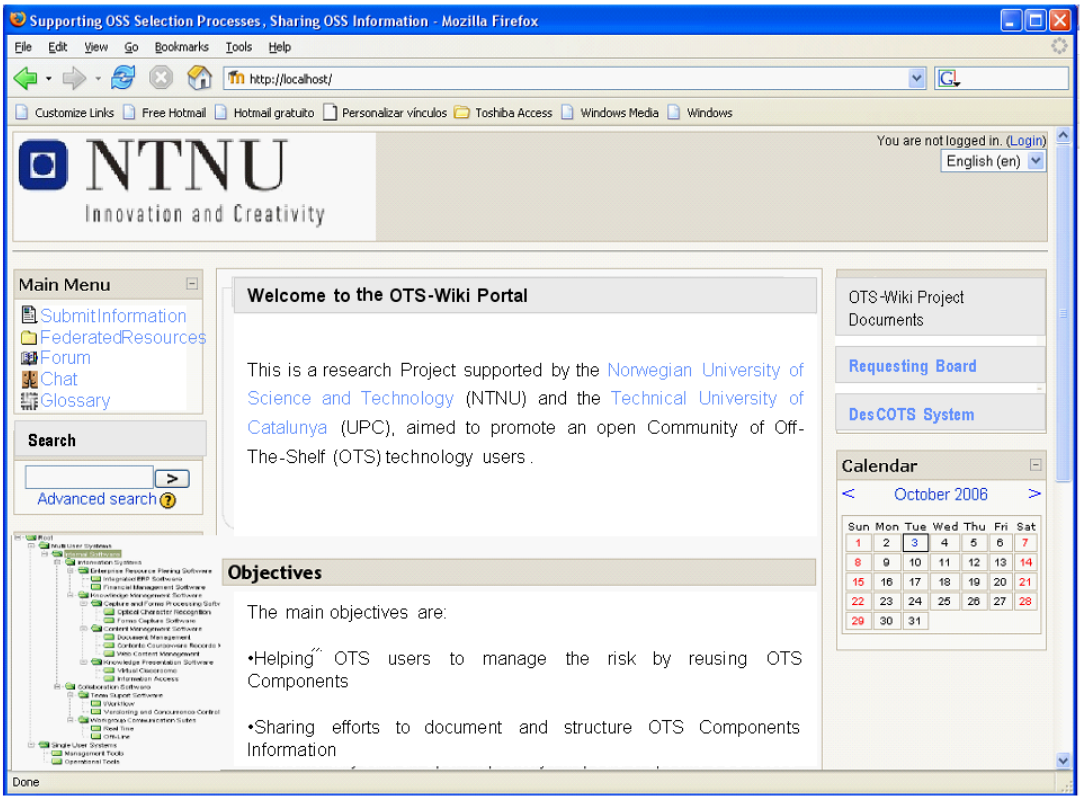

Figure 5.2: Suggested portal start page

5.3 Usability

As with all web portals, usability is very important in OTS-Wiki. In today's great variety of web portals and communities, the ones that survive are those that keep their users coming back over and over again. To grow and maintain a solid group of users, the portal must be perceived as easy to use. Usability is a vast term. The literature about usability contains many different definitions.

Wikipedia defines usability as follows[2]: "Usability is a term used to denote the ease with which people can employ a particular tool or other human-made object in order to achieve a particular goal. Usability can also refer to the methods of measuring usability and the study of the principles behind an object's perceived efficiency or elegance."

Shneiderman and Plaisant identifies five usability measures: time to learn, speed of performance, rate of errors by users, retention over time, and subjective satisfaction. The ISO 9241-11 standard identifies three aspects of usability: effectiveness, efficiency and satisfaction [Hornbæk and Law, 2007].

5.3.1 Usability principles

Usability is to a certain degree a subjective matter, and may vary between different individuals or groups of users. Gould and Lewis [Gould and Lewis, 1985] recommends three principles of design to foster usability of a computer system.

Early focus on users and tasks
It is important to find out who the users are, and the type of work these users are going to accomplish. This means *understanding* potential users, not only *identifying* them. The developers should also be in direct contact with potential users such as interviews and discussions. The user contact should be initialized before the system is designed.

Emiprical measurement
Users should be involved with testing of prototypes during development, so that their reaction and to the system could be observed and analyzed. It is important that the system is tested by the user, not demonstrated to them. A passive demonstration could result in misleading conclusions when analysing their reaction to it.

Iterative design
The problems found during testing should be fixed, and this is best done when performing iterative design and development.

5.3.2 Usability in the OTS-Wiki

The theory of usability factors related to OTS-Wiki may be summed up as follows:

Look and feel
The use of balanced colouring, styles, and images. These are important elements to give the user of the portal the impression of a credible and reliable source of information.

Intuitive navigation
Site navigation and user activities use well-proven patterns to make the user recognise

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

in the system and use it as it is supposed to be used (use of well-known keywords, icons, structures, and navigation rules).

Effective use
Easy and fast navigation, using a minimum number of clicks, without reaching "dead-ends".

Balanced data presentation
Presenting a balanced amount of data in a structural fashion.

User guidance
Available help functions and tips for optimal use of the site.

Eyecandy
"The little extra", design or functions, to impress the users and make them return over and over again.

5.4 Enabling Technology - The Java Platform

There exist many technologies suited for developing web-based applications. This section describes the Java approach, using Java technology as the foundation for developing the OTS-Wiki.

5.4.1 Java Introduction

Java[3] is an object-oriented programming language developed by Sun Microsystems. The first version was released in 1995, and has since then grown rapidly in both popularity and quality. *Portability*, *built-in networking support*, and *secure remote execution* were some of the main goals of creating the Java language. Platform independence, the idea of "Write Once, Run Anywhere" (WORA), has been one of the main features supporting these goals.

Java source code is compiled into bytecode (Java bytecode) that contains simplified machine instructions specific to the Java platform. The bytecode is then run on a Java Virtual Machine (JVM) that is written in native code on the host machine. Platform independence is achieved by implementing a JVM for each platform that need to run Java applications, making the same source code run on a number of different platforms[4].

Another important feature of the Java language is the automatic garbage collection. This leaves the work of memory management, garbage collection, and handling object lifecycles to the Java platform.

Since the release of version 2 of Java (Java 2), the technology was divided into three configurations supporting different types of platforms. The Java Standard Edition (Java SE) is the main configuration suitable for most common uses of Java. The Java 2 Enterprise Edition (Java EE) supports development of distributed, mulitier enterprise applications. The Java Micro Edition (Java ME) is a stripped down configuration aimed at limited and mobile platforms.

One of the main criticism of Java has been the perfomance issue. In the beginning, Java applications were perceived as slower than applications written in natively compiled languages, such as C or C++. Lately, the performance of Java has increased substantially, and the

[3] http://java.sun.com/
[4] http://en.wikipedia.org/wiki/Java_(programming_language)

difference between native compilers and Java compilers is no longer regarded as significant [Heiss, 2007]. Also, the look and feel of Java graphical user interfaces (using the Swing toolkit) has received some criticism. The difference in appearence between Java and native graphical interfaces has been regarded as significantly large. Some criticism also points out that Java's primitive data types are not objects, and thus, Java is not considered to be a pure object-oriented language.

The latest release of the Java Platform is the Java SE 6 (1.6.0).

5.4.2 Web Development in Java

One of the main goals of the Java technology was the built-in networking support. From there, the step is short to including web development frameworks and standards to the Java platform. Fundamentally, dynamic web content is supported by Java through the *Java Servlet API*[5]. A servlet is a Java object that handles HTTP requests to generate HTML or XML responses to a client web browser. Servlets need to run inside a web container to provide content to remote clients. The most common web container enabling servlets is *Apache Tomcat*[6]. Other popular web containers include *Jetty*, *Geronimo Application Server*, *IBM WebSphere*, *JBoss*, *BEA WebLogic*, *Borland Enterprise Server*, and *Oracle Application Server*.

A servlet is a cumbersome way to generate web content. JavaServer Pages (JSP) is a Java technology that enables dynamic scripts and Java code to be embedded directly into static HTML documents to generate dynamic responses to web requests[7]. The developer then do not have to worry about how to write the servlet code. When executed, a JSP is compiled into a servlet by a JSP compiler. Thus, JSP can be viewed as a high-level abstraction of the servlet API. Both servlets and JSP were originally developed by Sun Microsystems. Today, JSP is included in the Java Enterprise Edition.

5.4.3 Java Frameworks

JSP and servlets serve as the foundation for Java web development. To ease the developer's job of developing large, multiuser systems, there exist a number of frameworks suitable for web-based applications. Here, we will describe some important Java frameworks and how they can be used in the development of OTS-Wiki.

JavaServer Faces (JSF)

JSF[8] is a web application framework for simplified development of user interfaces (UI) for JavaServer/Java EE applications. The framework provides a set of APIs for representing reusable UI components and managing their state, handling events and input validation, defining page navigation, and supporting internationalization and accessibility[9]. JSF also provides wires from client-generated events to server-side event handlers/controllers and connections between the UI components and the application's data source. JSF adds a custom JSP tag library for expressing JSF interfaces within a JSP page.

Hibernate

Hibernate[10] is an object-relational mapping (ORM) solution for Java and .NET. The

[5] http://en.wikipedia.org/wiki/Servlet
[6] http://tomcat.apache.org/
[7] http://en.wikipedia.org/wiki/JavaServer_Pages
[8] http://java.sun.com/javaee/javaserverfaces/
[9] http://en.wikipedia.org/wiki/JavaServer_Faces
[10] http://www.hibernate.org/

framework provides mapping between an object-oriented domain model to a traditional relational database[11]. The goal is to relieve the developer from time-consuming data persistence-related programming tasks, such as saving, updating, deleting, and querying objects. Hibernate allows the developer to express queries in its own portable SQL extension (HQL), as well as in native SQL, or with an object-oriented Criteria and Example API. This makes Hibernate a powerful and flexible object-persistance framework suited for all non-trivial multitier applications [Hemrajani, 2006]. Hibernate is released under the *GNU Lesser General Public License (LGPL)* and may be used in both commercial and open source projects.

Spring

Spring[12] is an open source application framework for Java. It has gained popularity in the Java community as an alternative and replacement for the Enterprise JavaBean (EJB) model[13]. Spring provides a number of frameworks and best-practice solutions for building Java applications, especially for web-based applications on top of the Java EE platform [Hemrajani, 2006]. One of the main ideas is the use of configuration over system-specific coding for reusable tasks. The key features of Spring are:

- Configuration management based on JavaBeans, applying *Inversion-of-Control* (IoC) principles, specifically using the *Dependency Injection* technique. This aims to reduce dependencies of components on specific implementations of other components.
- A global core bean factory.
- Generic abstraction layer for database transaction management.
- Built-in generic strategies for JTA and a single JDBC DataSource. This removes the dependency on a Java EE environment for transaction support.
- Integration with persistence frameworks such as Hibernate, JDO and iBATIS.
- Model-Vew-Controller (MVC) web application framework, supporting many technologies for generating views, including JSP, FreeMarker, Velocity, Tiles, iText, and POI.
- Extensive aspect-oriented programming framework to provide services such as transaction management. As with the Inversion-of-Control parts of the system, this aims to improve the modularity of systems created using the framework.

The Spring Framework is licensed under the terms of the Apache License, Version 2.0.

Apache Struts

Apache Struts[14] is an open-source framework for developing Java EE Web applications. It uses and extends the Java Servlet API to encourage developers to adopt a MVC architecture[15]. The goal is to clearly separate the model (datalayer tasks) from the view (user interface), and the controller (mediator between the model and the view). Struts provides the controller (a servlet known as ActionServlet) and facilitates the writing of templates for the view or presentation layer (using JSP). The web application programmer is responsible for writing the model code, and for creating a central configuration file *struts-config.xml* which binds together model, view and controller. In December 2005, Struts joined forces with another popular Java EE framework, WebWork. The merge of Struts and WebWork is known as Struts 2. Apache Struts is licensed under the terms of the Apache License, Version 2.0.

[11] http://en.wikipedia.org/wiki/Hibernate_(Java)
[12] http://www.springframework.org/
[13] http://en.wikipedia.org/wiki/Spring_framework
[14] http://struts.apache.org/
[15] http://en.wikipedia.org/wiki/Struts

Apache Tapestry
Apache Tapestry[16] is a Java-based web programming toolkit that uses XML to implement applications in accordance with the MVC design pattern[17]. Its component-based architecture provides strong bindings between the elements on a web page and the underlying code to facilitate fast and straightforward development. Apache Tapestry is licensed under the terms of the Apache License, Version 2.0.

Apache Lucene
Apache Lucene[18] is an open source search engine library written in Java[19]. It provides textual searches in an application's domain model or documents using indexing. Lucene is used in a wide range of applications and portals, and is well known for its flexibility, scalability, and accuracy. It is also ported to other programming languages such as Perl, C#, C++, Python, Ruby, and PHP. Apache Lucene is released under the Apache Software License.

5.4.4 Security in Java Web Applications

Security is an important issue in all software applications. In web applications, security challenges are even more important. Evidence shows that perhaps as many as sixty percent of attacks on enterprise web applications are facilitated by exploitable vulnerabilities present in the source code [Lebanidze, 2006]. On the other hand, [Grossman, 2005] points out that Pareto's Principle or the 80/20 rule, suggesting that 20% of the defects causes 80% of the problems, applies in web application projects. Further, he suggests a number of countermeasures to increase security without touching the source code. Since most of the countermeasures are applied at the web server level, they are beyond the scope of this report to discuss in detail. Here we focus on what security issues can be resolved in the application code, specifically in Java.

When evaluating the security of web applications the core security services collectively known as *CI4A* (Confidentiality, Integrity, Authentication, Authorization, Availability, and Accountability) [Lebanidze, 2006] are mentioned. Confidentiality is concerned with the privacy of information that passes through or is stored inside the web application. Integrity ensures that the data used is free from modification. Authentication addresses verification of identities. Authentication can also be thought of in the context of source integrity. Authorization focuses on access rights to various application subsystems, functionality, and data. Availability, an often ignored aspect of security, is nevertheless an important metric for the security posture of the web application. Many attacks that compromise application availability exploit coding mistakes introduced at the application source level that could have been easily avoided. Non-repudiation addresses the need to prove that a certain action has been taken by an identity without plausible deniability. Accountability, tied with non-repudiation, allows holding people accountable for their actions.

A major task, often neglected as trivial by developers, is the user authentication and authorization. Today, enterprise applications often include a number of independent or semi-dependent web applications that users access daily. Handling user credentials in these heterogeneous environments is a great challenge. Single sign-on (SSO) is a session/user authentication process that allows a user to provide her credentials once in order to access multiple applications[20].

[16] http://tapestry.apache.org/
[17] http://en.wikipedia.org/wiki/Apache_Tapestry
[18] http://lucene.apache.org/java/docs/
[19] http://en.wikipedia.org/wiki/Lucene
[20] http://en.wikipedia.org/wiki/Single_sign-on

The single sign-on authenticates the user to access all the applications she has been authorized to access. It eliminates future authenticaton requests when the user switches applications during that particular session. Web Single sign-on works strictly with applications accessed with a web browser. The request to access a web resource is intercepted either by a component in the web server, or by the application itself. Unauthenticated users are diverted to an authenticaton service and returned only after a successful authentication.

Here we briefly describe some of the frameworks and technologies related to web security and single sign-on in Java.

Java Authentication and Authorization Service (JAAS)
JAAS[21] s a set of APIs that enable services to authenticate and enforce access controls upon users. It implements a Java technology version of the standard Pluggable Authentication Module (PAM) framework, and supports user-based authorization. JAAS is created by Sun Microsystems and is included in Java Runtime Environment 1.4 and later.

JA-SIG Central Authentication Service (CAS)
CAS[22] CAS is an authentication system originally created by Yale University to provide a trusted way for an application to authenticate a user. It is implemented as an open source Java server component and supports a library of clients for Java, PHP, Perl, Apache, uPortal, and others. CAS serves as a foundation for several other security frameworks and SSO solutions.

Acegi Security
Acegi Security[23] is a Java-based security framework supporting authentication, authorization, instance-based access control, channel security and human user detection. It is especially well coupled with the Spring framework, and serves as a basis for CAS.

Java Open Single Sign-On (JOSSO)
JOSSO[24] is an open source Java EE-based SSO infrastructure aimed to provide a solution for centralized platform neutral user authentication. It is based on JAAS and uses Struts and JSP, as well as web services (Axis) as the distributed infrastructure. The use of web services allows integration of non-Java applications (i.e. ASP.NET or PHP).

5.4.5 Java Development Tools

There exist a number of development platforms and Integrated Development Environments (IDE) for Java. An IDE includes most of the tools needed to write, debug, and run an application. Below we descibe some of the most common Java IDEs and some other development tools to ease the Java development.

Eclipse
Eclipse[25] is an open source development platform and IDE comprised of extensible frameworks, tools and runtimes for building, deploying and managing software across the lifecycle. Eclipse has become the most popular open source IDE due to the extensive base of plugins and third party applications, making almost every part of application development possible within Eclipse [Hemrajani, 2006].

[21] http://java.sun.com/products/jaas/
[22] http://www.ja-sig.org/products/cas/
[23] http://www.acegisecurity.org/
[24] http://www.josso.org/
[25] http://www.eclipse.org/

NetBeans
NetBeans[26] is an open source Java IDE owned by Sun Micosystems. It provides tools needed to create cross-platform desktop, enterprise, web and mobile applications. NetBeans runs on Windows, Linux, MacOS, as well as Solaris.

JDeveloper
JDeveloper[27] is a free IDE, made by Oracle, with end-to-end support for modeling, developing, debugging, optimizing, and deploying applications and Web services. It relies mainly on the Oracle platform and supports multiple technologies and programming languages, such as Java, XML, SQL and PL/SQL, HTML, JavaScript, BPEL, and PHP.

JBuilder
JBuilder[28] is a Java IDE from Borland/CodeGear. It has won several awards as the most powerful IDE for professional Java Programming. JBuilder is available commercially, as well as a free limited version for beginners.

JUnit
JUnit[29] JUnit is an open source regression testing framework written by Erich Gamma and Kent Beck. It has become the de-facto standard of performing unit tests in Java. Unit testing the source code is regarded a mandatory task in modern software development. JUnit is integrated into a number of frameworks and IDEs "out of the box", and is also available for a number of other programming languages.

Apache Ant
Ant[30] is an open source software tool written in Java for automating software build processes. Ant uses XML to decribe the build process and its dependencies.

Apache Maven
Maven[31] Maven is an open source software project management and comprehension tool. Based on the concept of a project object model (POM), Maven can manage a project's build, reporting and documentation from a central piece of information. Similar to Ant, Maven uses XML to describe the build process. Maven is often regarded as Ant's successor, but both projects coexist and are widely used in the Java software industry.

Subversion
Subversion[32] is an open source version control system for managing source code and versions in projects with multiple developers. Subversion is not Java-specific, but the tool is still important in any development project.

5.4.6 Alternative Web Development Platforms

Here, we briefly discuss some alternative web development platforms and technologies that may perform the same tasks as the Java platform.

[26] http://www.netbeans.org/
[27] http://www.oracle.com/technology/products/jdev/
[28] http://www.codegear.com/products/jbuilder
[29] http://www.junit.org/
[30] http://ant.apache.org/
[31] http://maven.apache.org/
[32] http://subversion.tigris.org/

PHP
PHP[33] (recursive acronym for "PHP: Hypertext Preprocessor") is an open source general-purpose scripting language that is especially suited for Web development and can be embedded into HTML. PHP generally runs on a web server, taking PHP code as its input and creating Web pages as output. A number of commercial and open source web servers can run PHP, and it is compatible with most common database systems. Lately, the LAMP platform has become popular in the open source web industry. PHP is commonly used as the P in this bundle alongside Linux, Apache and MySQL. PHP as a scripting language can be compared to JSP, but combined with frameworks, such as CakePHP, PRADO, Symfony, or the Zend Framework, PHP may serve as a comprehensive web development platform.

ASP.NET
ASP.NET[34] is Microsoft's web development platform. It can be used to build dynamic web sites, web applications and XML web services. It is part of Microsoft's .NET platform and is the successor to Microsoft's Active Server Pages (ASP) technology. ASP.NET is built on the Common Language Runtime (CLR), meaning programmers can write ASP.NET code using any Microsoft .NET language (C#, C++/CLI, Visual Basic, J#, IronPython, IronRuby, and others).

Ruby on Rails
Ruby on Rails[35] (RoR) is an open source project written in the Ruby programming language and applications using the Rails framework are developed using the Model-View-Controller design pattern. Rails provides out-of-the-box scaffolding, a skeleton code framework which can quickly construct most of the logic and views needed for a basic website. RoR has become popular due to its ability to enable quick and easy development of database-driven web sites.

5.5 The Cosiportal Project

The student Per Kristian Schanke at the Norwegian University of Science and Technology (NTNU) is working on a project to provide a portal for companies to manage projects after they are released as Open Source. The Cosiportal Project will be hosted at the same server as OTS-Wiki, and Schanke is currently responsible for administration and maintenance of all the server applications, including the ones related to OTS-Wiki.

5.5.1 Background

Open Source is a field in Computer Science that grows steadily. With the success of Linux, Mozilla and other projects is there a surge in interest among companies to release their commodity software as Open Source. There are different reasons for this; publicity and getting others to improve on code that is not business critical are some of the major reasons. Another reasons worth to mention is the fact that more and more companies use Open Source components in their development. To be able to sell this software, they usually need to follow the license that the components is released under and offer their finished product as Open Source.

[33] http://www.php.net/
[34] http://www.asp.net/
[35] http://www.rubyonrails.org/

5.5.2 The project

Even though companies release software as Open Source, they want to keep some control of the further development. The project aims to provide the portal to enable companies to release their source code as Open Source and still keep control. The mission of the project is to develop a portal companies can install on their web servers, to get all the tools they need to enable external users to contribute to the project. The components used in the portal are Open Source, and these are tied together into a package which should be easy to install and upgrade. The portal do currently contain a SVN-repository, a mailing-list and a bug-tracker. It will eventually also contain a forum.

Chapter 6

Requirements Specification

This chapter presents the functional and non-functional requirements of the OTS-Wiki portal. A use case specification is used to show the actions and interactions of the system's actors, and to describe the most important functionality.

6.1 Functional Requirements

This section presents the functional requirements of OTS-Wiki. The goal is to describe the functions and actions related to the actors in the system. Firstly, we present the actors and their concerns. Lastly, we present an overview of the specified use cases. Each use case is given a priority. This is done to guide the development and implementation of the system. The highest prioritized use cases are implemented early in the incremental development process, and the lower prioritized functions are implemented later. The complete use case specification is presented in Appendix A.

The specified use cases represent the basic functionality we plan to implement in this project. Due to time issues, all the features and functions mentioned in Section 2.1 are beyond the scope of this thesis to implement. Our goal is to implement enough functionality to begin populating and using the portal, and to evaluate the work done to guide the further evolution of OTS-Wiki.

6.1.1 Actors

The use cases are specified using UML, as described by [Larman, 2001]. In UML, an actor is something or someone who supplies stimulus to the system. The primary actors interact directly with the system to achieve certain goals. The primary actors may be supported by supporting actors, such as external persons or systems that indirectly contribute to the goals. Often, the system stakeholder are modelled as actors. They do not interact directly with the system, but have interests in the primary actor's goals. Active actors provide input to the system, while passive actors act as targets of information or requests. Passive actors may also be activated by the system. Figure 6.1 shows the actors and their dependencies. The actors are described below.

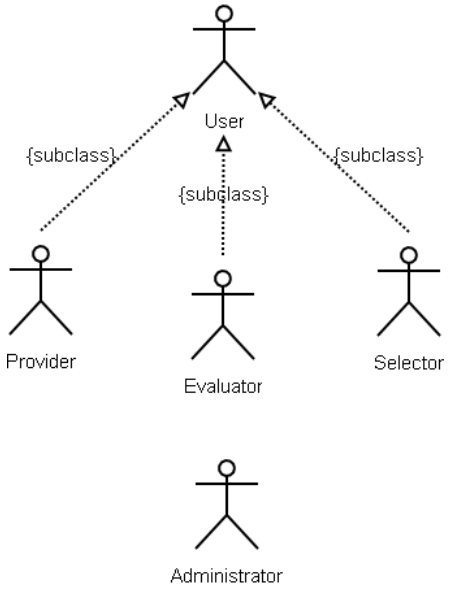

Figure 6.1: The system actors

User
The User is the superclass actor for all the end users in the system. The *Provider*, the *Evaluator*, and the *Selector* are subclasses of the User. In the use case specification, the User actor is used in cases where the user has not yet adopted one of the subclass roles. All general portal utilities, such as the forum, the chat rooms, and general read-only activities, are actions that belong to the User actor.

Provider
The Provider is the actor responsible for all the contributions not related to the evaluations of the components. Adding, editing, and requesting components are the core tasks of the Provider. Also adding resources and glossary items are important. The Provider plays the main role of populating the community's database. A user has to be registered to become a Provider.

Evaluator
The Evaluator is the actor that performs the evaluations of the components in the community. The evaluation can be done on several detail levels. A user has to be registered to become an Evaluator.

Selector
The Selector is the actor that is actively seeking candidate components and information. The Selector uses the system as a selection support tool to find and use one or more software components. A user do not have to be registered to become a Selector, but to get full selection support, registration is needed.

Administrator
The Administrator is the actor that administers and maintains the system. The Administrator is also a superuser with all rights in the system. User handling, role assignment, and statistics work are some of the tasks of the Administrator.

6.1.2 Use Case Overview

Table 6.1 presents an overview of the specified use cases. The use cases are prioritized to guide the implementation work. The complete use case specification is presented in Appendix A.

Table 6.1: Use Case Overview

ID	Name	Actor	Priority
UC1.1	Register	User	High
UC1.2	Login/Logout	User	High
UC1.3	Edit User Information	User	Medium
UC1.4	Make Forum Post	User	Medium
UC1.5	Chat	User	Low
UC1.6	Browse Content	User	High
UC1.7	Get Help	User	Low
UC2.1	Add Component	Provider	High
UC2.2	Edit Component	Provider	High
UC2.3	Request Component	Provider	High
UC2.4	Add Resource	Provider	Medium
UC2.5	Edit Glossary	Provider	Medium
UC3.1	Rate Component	Evaluator	High
UC3.2	Comment Component	Evaluator	High
UC3.3	Provide Detailed Evaluation	Evaluator	High
UC4.1	Textual Search	Selector	High
UC4.2	Advanced Search	Selector	High
UC5.1	Log User Activity	Administrator	Medium
UC5.2	Handle Requests	Administrator	Low
UC5.3	View Statistics	Administrator	Low
UC5.4	Moderate Content	Administrator	Medium

6.2 Non-Functional Requirements

The book Software Architecture in Practice [Bass et al., 2003] defines a set of quality attributes related to the quality of the system.

Availability

Definition of availability: "Availability is concerned with system failure and its associate consequences. A system failure occurs when the system no longer delivers a service consistent with its specification" [Bass et al., 2003].

Availability is an important factor for the success of OTS-Wiki. This means that the portal should handle all types of input from the user without producing a system failure, but instead give the user informative feedback.

Modifiability

Definition of modifiability: "Modifiability is about the cost of change (..) 1. What can change (the artefact)? (..) 2. When is the change made and who makes it (the environment)?" [Bass et al., 2003]

OTS-Wiki is in an early development phase, which means that a lot of new functionality and improvements are going to be made to it. The modifiability is naturally crucial when developing the existing solution further, to avoid spending unnecessary time because of complex design.

Performance

Definition of performance: "Performance is about timing. Events (interrupts, messages, requests from users, or the passage of time) occur, and the system must respond to them" [Bass et al., 2003].

OTS-Wiki should react in real-time to avoid frustrated users. This means waiting time no longer than it usually takes to load a web site.

Security

Definition of security: "Security is the measure of the system's ability to resist unauthorized usage while still providing its services to legitimate users" [Bass et al., 2003].

Users should only be able to get access to the functionality their user level gives them authorization to.

Testability

Definition of testability: "Software testability refers to the ease with which software can be made to demonstrate its faults through (typically execution-based) testing" [Bass et al., 2003].

OTS-Wiki will use unit testing, using JUnit, to eliminate logical errors in the source code. Unit testing is easily integrated in most IDEs and is supported by most common web development platforms.

Usability

Definition of usability: "Usability is concerned with how easy it is for the user to accomplish a desired task and the kind of user support the system provides" [Bass et al., 2003].

Usability is one of the most important success factors when releasing OTS-Wiki. Without satisfactory usability, people will probably not continue to use the portal after have tried it once. Usability means intuitive user-interface, mostly self explaining functionality, efficient use of the available functionality etc. More about usability in Section 5.3 in Chapter 2.

6.2.1 The non-functional requirements

Table 6.2 presents the non-functional requirements of the system. Each requirement is identified and linked to a specific quality attribute.

Table 6.2: Non-functional requirements

ID	Quality Attribute	Description
NFR1	Availability	The system should handle all types of user input without going down.
NFR2	Availability	The system should be available for use in the most common web browsers.
NFR3	Usability	A user should be able to perform search and browse the content of the OTS-Wiki without needing any instruction.
NFR4	Performance	The system should handle user request fast enough for real time interaction.

Chapter 7

System Architecture

This chapter will describe the overall architecture of OTS-Wiki. This includes a presentation of the system stakeholders and their concerns. We also look at the most important quality attributes and present some tactics to manage the system quality. Further we present the Model-View-Controller (MVC) architectural pattern that the portal is based upon, and describe the data model that serves as the foundation of the model-part (datalayer) of the architecture. Lastly, we present the technological choices for the portal.

7.1 System Stakeholders

Table 7.1 presents the system stakekholders and their concerns.

Table 7.1: System stakeholders and their concerns

ID	**Stakeholder**	**Concern**
S1	IDI and NTNU, represented by Carl-Fredrik Sørensen and Reidar Conradi	Implementation to support research questions related to selection, search and evaluation
S2	Technical University of Catalunya, represented by Claudia Ayala	Implementation to support research questions related to Claudia Ayala's PhD work
S3	The OTS community	Means for simplified and open support for selection and evaluation of OTS components
S4	The open source community	Development of OSS components and further development of OTS-Wiki
S5	The developers	Specification and development of the system

7.2 Architectural Patterns

Here, we present the most important architectural patterns [Bass et al., 2003] to use in OTS-Wiki, the *Client-Server*, and the *Model-View-Controller* pattern. Lastly, we describe a specific tactic for writing controllers in the MVC-structure.

7.2.1 Client-Server

The client-server pattern is the base architectural pattern used in most web applications. OTS-Wiki will use the standard client-server configuration with a central web server serving clients on the web accessing the portal using their web browser. The web server will be co-localized with the database server, ensuring high performance and maintainability. The server will initially be hosted at IDI, NTNU.

7.2.2 Model-View-Controller (MVC)

We have chosen a Model-View-Controller (MVC) architectural pattern as the foundation for OTS-Wiki. The MVC-approach is well proven and widely used in web applications. A number of suitable frameworks and best practice solutions and patterns for web development are based on MVC [Hemrajani, 2006, Bass et al., 2003]. One of the main principles of the MVC pattern is *separation of concerns*.

In web applications, which often present lots of data to the user, the separation of the data (*model*) and the user interface (*view*) is often preferable. In this way, changes to the user interface do not affect the data handling, and a reorganizing of the data is possible without needing to change the user interface. The MVC pattern solves this problem by decoupling data access and business logic from data presentation and user interaction, by introducing an intermediate component: the *controller*.

Separating concerns means that changing one part/component do not affect the other parts of the system significantly. This is positive for the system's modifiability. Also, this makes possible testing of each component or layer separately, increasing testability.

Figure 7.1 depicts the MVC structure of OTS-Wiki. The view is represented by JSP-pages, using Java and basic web technologies, such as HTML and CSS, to present the content to the user. The model consists of base Java objects and Data Access Objects (DAO) accessing the database. This layer also includes mappings and configurations for data persistence and transaction support (Using Hibernate, not depicted in the diagram). The controller classes are described in detail below.

7.2.3 POJO Controllers

In Java web applications, the controller is often represented by a servlet that receives requests from the user interface (JSP-pages), fetches the correct information from the data-layer, and sends a response to a resulting JSP-page. The use of different MVC frameworks for Java, such as Struts, Spring, and JSF, makes possible the use of other controller techniques. Hand in hand with the growing popularity of lightweight/agile software engineering processes and methods, such as Scrum and eXtreme Programming, the agile mindset is also influencing the source code. An important trend is going back to using more and more *Plain Old Java Objects* (POJOs) [Richardson, 2006] in the business logic of multitier applications. Using this approach,

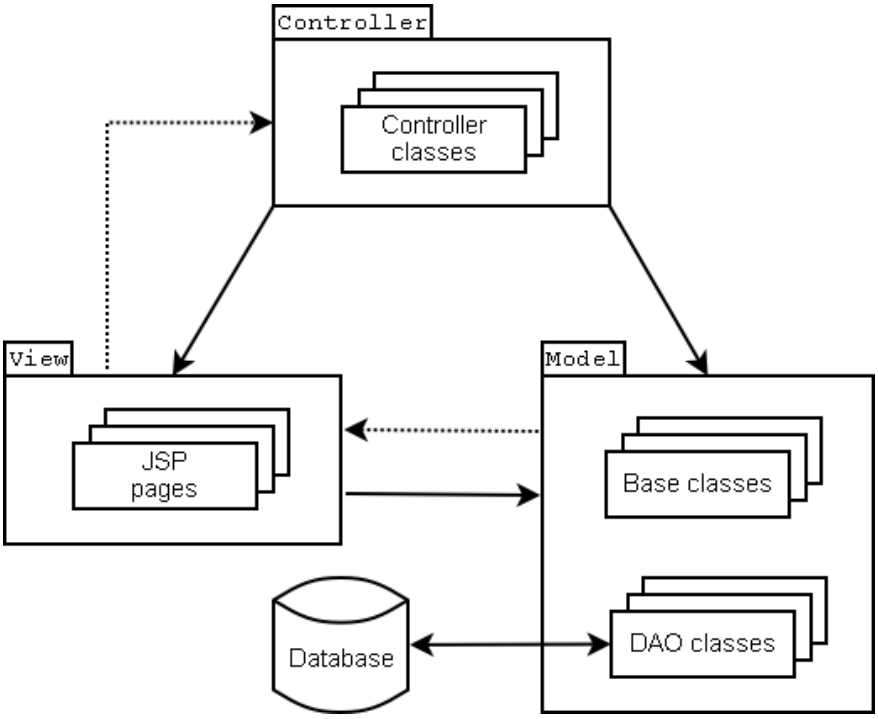

Figure 7.1: MVC structure

controllers may be written as POJOs to ease the development and simplify the adaptation of additional frameworks (Spring, Hibernate, JSF, and others).

We have chosen the POJO approach in the controller layer of OTS-Wiki. The functionality is provided to the view layer through standard Java methods (get and set). The controllers group the functionality logically to support the system's use cases. Each JSP-page uses one or more controller objects, which in turn use a set of base and DAO objects from the model layer to fetch and store the data. Figure 7.2 shows an example of the linking between the JSP-pages, the controller objects, and the data-layer.

7.3 Data Model

Figure 7.3 depicts the system's data model. The model is represented as an normalized Entity Relationship Diagram (ERD) [Larman, 2001] using the crows foot notation.

The system's main entity is the *Component*. A component may belong to one or more *Categories*, use one or more *Technologies*, and run on one or more *Platforms*. In addition, the components may be linked to one or more descriptive *Tag*. A component also has information about the *Vendor*, the *Versions*, the *Licenses*, and a set of additional *Component resources*. The other key entity is the *User*, which maintains a *Glossary*, the portal *News*, and a list of *Resources* relevant to the OTS community. The user also rates (*Rating*), comments (*Comment*), and evaluates (*Evaluation*) the components.

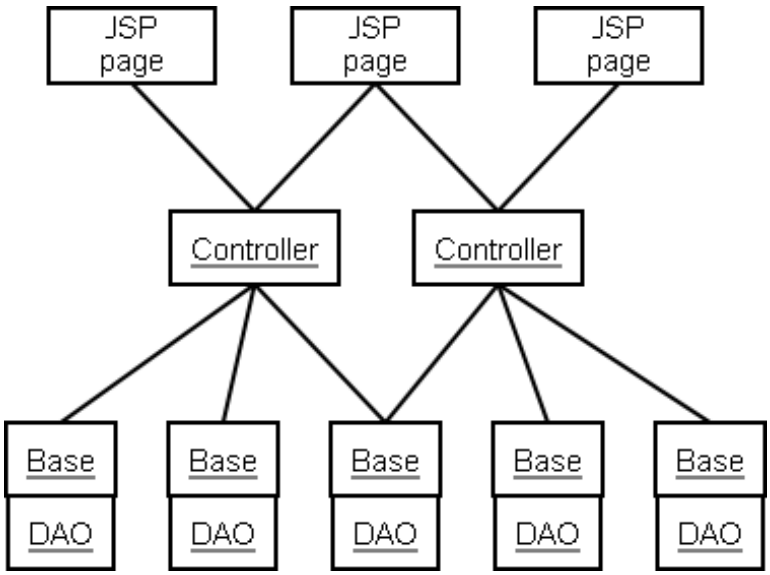

Figure 7.2: MVC example

7.4 Technology Choices

7.4.1 Java Platform

OTS-Wiki will be developed using the Java platform, as described in detail in Section 5.4. We have chosen Java because the platform offers complete and good solutions for developing, testing, and running web applications. The availability and quality of open source tools, such as Hibernate, Eclipse, JUnit, and Tomcat made the choice easy. We chose to buy licenses of MyEclipse[1], a commercial web development suite for Eclipse. MyEclipse simplifies integration and use a number of common plug-ins and time-saving enhancements in Eclipse. The use of open source tools and platforms has been an important principle in this project, and besides MyEclipse, all the tools used are open source. Due to complexity and limited implementation time, we have decided to use a minimum of Java frameworks. Hibernate is the only framework we use now, but more frameworks, such as Spring, Acegi Security, Apache Lucene, and JSF may benefit the system in the future. Table 7.2 presents an overview of the technologies used.

[1] http://www.myeclipseide.com/

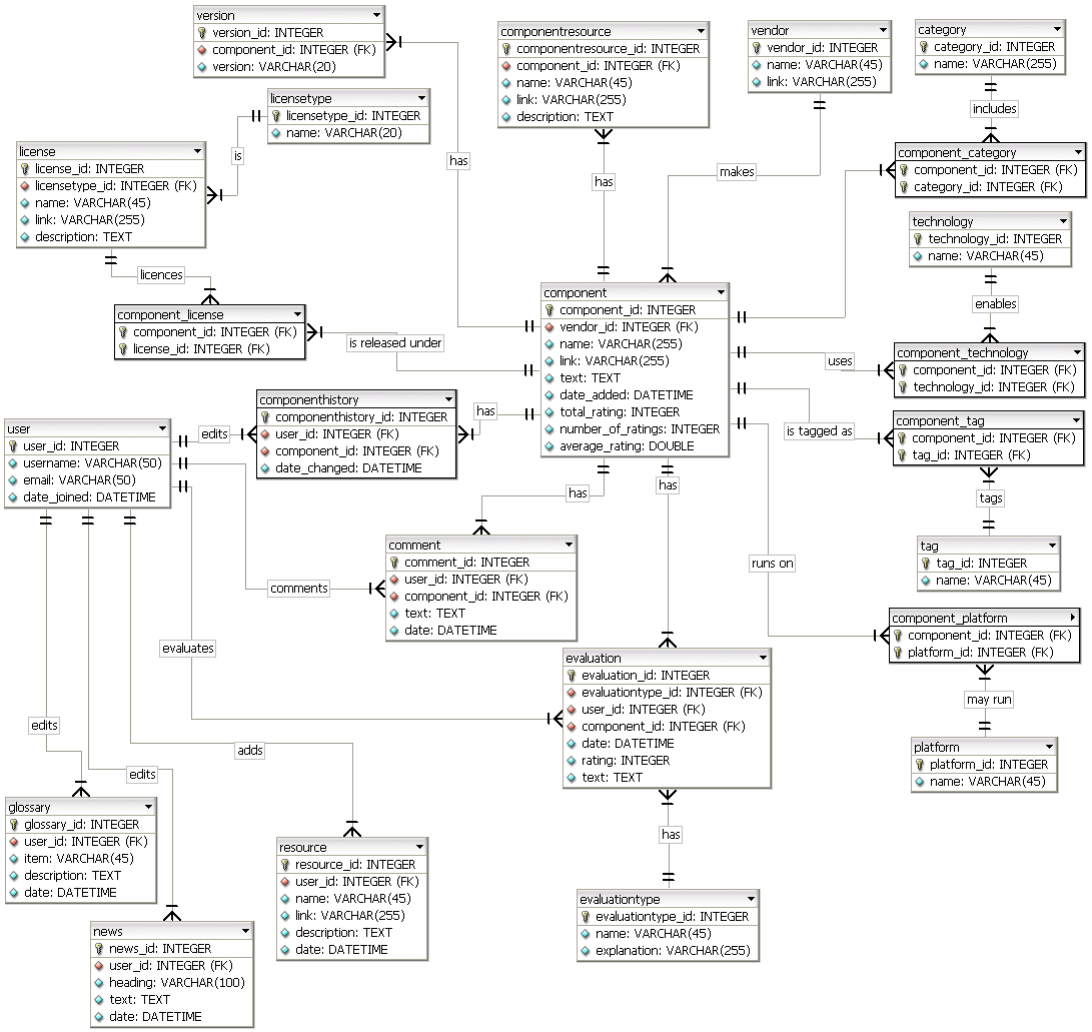

Figure 7.3: The data model

Table 7.2: Technologies used in OTS-Wiki

Part	**Technology**
Programming language	Java (Java SE 1.5 and Java EE 1.4)
Object persistence	Hibernate 3.1.3
Graphical user interface	HTML, CSS, and JSP
Web container	Apache Tomcat 5.5.20
Database	MySQL 5.0.24
Development environment	Eclipse 3.2 and MyEclipse 5.5 M1
Testing harness	JUnit 4.1
Modeling tools	DBDesigner 4 and Dia 0.95

Part IV

Evaluation

Chapter 8

Evaluation

This chapter documents and evaluates the implementation done in this thesis. We describe what is implemented, and how. We look at the efforts for ensuring usability, and discuss problems and lessons learned during the development.

8.1 The Implementation

We have developed OTS-Wiki with the specified architecture and use cases in mind. The implementation of the entire set of features, as mentioned in the vision (Section 2.1), well exceeds the scope of this thesis. The implementation mainly covers the basic functionality needed to add, find, edit, and evaluate OTS components. These are the functions needed to start using the portal and also to get users to form an opinion of the concept and the evolution of OTS-Wiki. In this section, we describe in detail what is implemented so far. We discuss the implemented use cases, the component metadata, and the user handling.

8.1.1 Implemented Use Cases

Table 8.1 shows which use cases have been fully implemented (FI), partly implemented (PI), or not implemented (NI). A comment is provided to briefly explain the partly implementations and the use cases not implemented.

The main parts that are not implemented are the administration tasks and the user profiling. These will be important tasks in further work.

Table 8.1: Use Case Implementation Overview

Use Case	FI	PI	NI	Comment
User Actions				
UC1.1 Register	X			
UC1.2 Login/Logout	X			
UC1.3 Edit User Information			X	Not prioritized due to time issues
UC1.4 Make Forum Post			X	Not prioritized due to time issues
UC1.5 Chat			X	Not prioritized due to time issues
UC1.6 Browse Content		X		Source code not available to the user
UC1.7 Get Help			X	Not prioritized due to time issues
Provider Actions				
UC2.1 Add Component	X			
UC2.2 Edit Component	X			
UC2.3 Request Component			X	Not prioritized due to time issues
UC2.4 Add Resource	X			
UC2.5 Edit Glossary	X			
Evaluator Actions				
UC3.1 Rate Component	X			
UC3.2 Comment Component	X			
UC3.3 Provide Detailed Evaluation	X			
Selector Actions				
UC4.1 Textual Search	X			No search framework/indexing used
UC4.2 Advanced Search	X			
Administrator Actions				
UC5.1 Log User Activity			X	Not prioritized due to time issues
UC5.2 Handle Requests			X	Not prioritized due to time issues
UC5.3 View Statistics			X	Not prioritized due to time issues
UC5.4 Moderate Content			X	Not prioritized due to time issues

8.1.2 Implemented Component Metadata

Based on the component metadata descibed in Section 2.1.1, we have implemented a simple classification and description of the components in OTS-Wiki. This work is not final, but the base implementation is extensible with more metadata.

Domain Function
The domain function is mainly represented by a categorization of the components. A component may belong to zero or more categories. The base categories are: *Clustering, Database, Desktop, Development, Enterprise, Financial, Games, Hardware, Multimedia, Networking, Security, Storage, SysAdmin*, and *VoIP*.

Quality
No metadata for quality measurements have been implemented. The integration of DesCOTS (Section 3.3) may provide means of determing component quality, but this has not been prioritized in this thesis.

Technical Properties
The technical properties implemented are: *Technology*, *Platform*, and *Tag*. The technology encapsluates the programming languages the component is developed in and suitable for. The platform describes the component's operating system requirements. Lastly, the tag is a set of keywords describing the component. The technical properties, as well as the categorization, are made searchable in the portal, enabling detailed component selection support.

Non-Technical Properties
The component's non-technical information includes the following properties:

- The name of the component
- A textual description
- A link to the component's website
- The name of the vendor
- The component's licensing information
- The component's versions

These properties are editable by the users of the portal, enabling Wiki-style collaboration.

Experience Properties
The evaluation part of the portal has been divided into three levels. The lowest level of evaluation is the *rating*. All users may rate a component (1-6) based on the general view of the component. The rating is anonymous and used to calculate average rating. The highest rated components are presented on the main page of the portal for quick access. The middle level of evaluation is the *comment*. This is a general comment to describe the user's overall impression of the component and its functionality. The highest level of evaluation includes both a rating and a comment aimed at distinct concerns or evaluation types: *Installation, Documentation, Integration*, and *Degree of Goal Achievement*. The installation evaluation describes and rates the effort of acquire and install the component. The documentation evaluation describes the state and accessibility of the component's documentation. The integration evaluation describes the effort of integrating the component with other components or systems. Lastly, the component is evaluated on how well it achieves the goals and requirements. The portal can easily be extended with additional evaluation types and concerns.

8.1.3 The User Database

The user management is implemented in cooperation with the Cosiportal Project. The implementation is based on the single sign-on (SSO) idea, enabling a user of one portal to automatically be registered on the other portal if needed. Ideally, a SSO framework, as descibed in Section 5.4.4, should be used, but this was not prioritized. In stead, both portals shared a common user table, storing the user credentials and contact information. A custom authentication controller was written in OTS-Wiki to access the user database in the Cosiportal to register and authenticate users. A copy of the user object is stored locally to enable user profiling specific to the needs of OTS-Wiki.

The user handling and profiling has not been prioritized in this thesis. This remains an important case for further work.

8.2 Usability Evaluation

One of the most important factors to succeed in competition with other web portals offering much of the same functionality is good usability. To ensure usability, it is important to perform tests. Usability testing is expensive, and the time has been limited when implementing the OTS-Wiki. We have therefore chosen to put a little extra effort in the design process to ensure a certain level of usability.

8.2.1 Design

Usability is defined as the ease of use of a certain tool. The design of OTS-Wiki is therefore an important usability issue.

When designing the user interface of OTS-Wiki, we have tried to keep the design as simple and clean as possible. The design is a standard three column layout with header and footer illustrated in Figure 8.1. The navigation menu is placed in the left column, the main in the middle, and other information such as component top-lists are placed in the right column. Most web-users should be familiar with this positioning. Further, the log-in functionality is located at the right side in the header, which is normal in many web-pages requiring user-authentication.

The central aspect of OTS-Wiki is component search. The search box providing this functionality is therefore placed at the top of the middle column, where it is easy accessible.

See Appendix B for more screenshots of the implemented portal.

8.2.2 Improving usability

The usability of OTS-Wiki is one concern where effort should be placed when developing the concept further. This involves usability testing to identify possible problem areas such as unintuitive navigation.

The user should be able to use the most common functionality in OTS-Wiki without the need of reading documentation. Still, OTS-Wiki needs some help functionality. This includes regular text documentation, a Frequently Asked Questions (FAQ) section, and wizards guiding the user through specific tasks such as component evaluations. We will discuss some of these issues further in Chapter 10, Further Work.

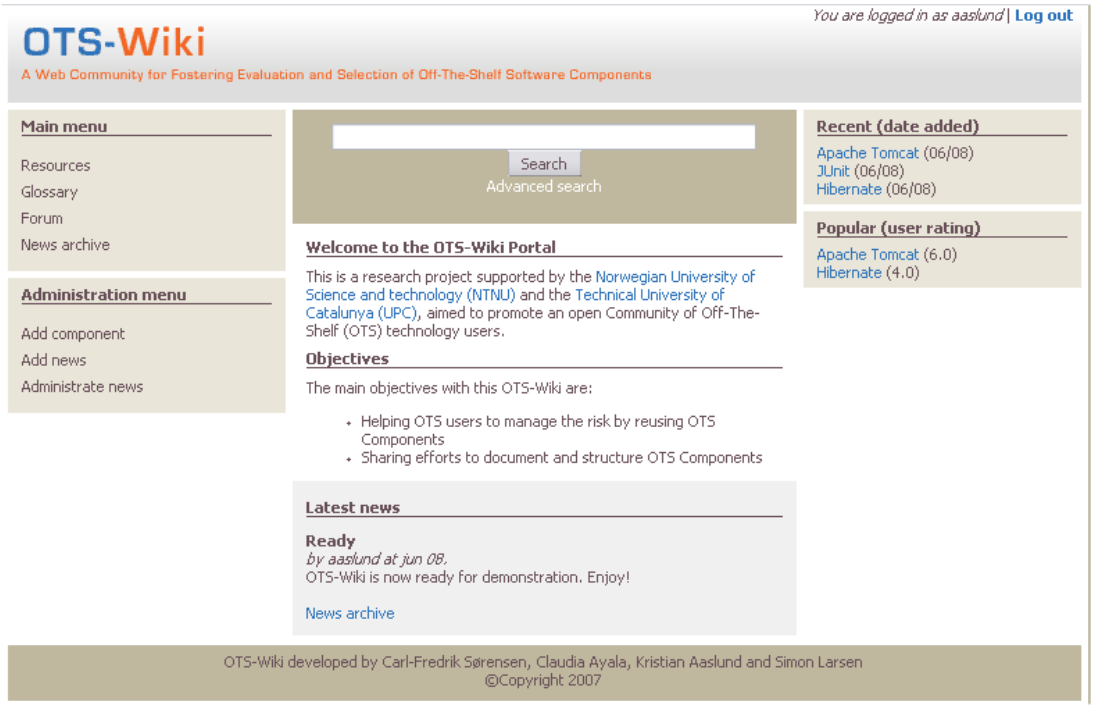

Figure 8.1: OTS-Wiki mainpage

8.3 Evaluating the Development Process

We have based the development process on an agile mindset, influenced by Scrum and XP (Section 3.6). A team of only two developers is too small to fully apply Scrum, but the principles of Scrum have still been important in the development process.

Collaborating development was made possible through the use of source code management and version control with Subversion. This removed the need of co-locating the developers at all times, and improved the efficiency of the implementation.

8.3.1 Front-End vs Back-End

In the initial phase of the implementation, we decided to divide the work into two parts: *front-end* and *back-end*. The front-end part consists of the view layer of the MVC architecture, including the design and implementation of the web pages using JSP, HTML, and CSS. The back-end part is a regular Java task, implementing the controller and model layers of the architecture. Here, modifiability and availability have been important focus areas, while usability has been the main focus of the front-end part.

Separating these tasks has made possible a more in-depth focus on the different layers of OTS-Wiki, increasing the quality of the work done in each layer. Also, the prior experience of the developers made this separation natural and easy. On one hand, the back-end developer makes the data accessible and editable, and on the other hand, the front-end developer presents the data to the user. The key to this work is a clean and understandable controller layer. A typical work flow may look like this:

1. The front-end developer implements a form in the user interface
2. The front-end developer requests a set of controller methods supporting the form actions
3. The back-end developer implements the controller methods and the needed model support (Data access)

The main disadvantage of this separation has been the lack on insight in the other developer's code and work. An important principle in XP is pair programming, where all the code is revised by both developers. This has proven to reduce error rate and increase the code quality [Hemrajani, 2006]. This taken into account, we believe that the separation was the most efficient way of conducting the implementation in this project.

8.3.2 Vertical Development

An important principle in agile methodology is *vertical development*. In a multitier application, such as the MVC structure of OTS-Wiki, each visible function in the user interface has roots down to the datalayer. Vertical development means developing a complete function, top-down or bottom-up, with DAO connectivity, controller methods, and user interface before starting on the next function.

8.3.3 Refactoring

Refactoring is important in XP. The principle of "start writing code, and refactor it later" has worked well in this project. We started off with an initial data model and a set of use cases to base the portal on. This differs slightly from the agile mindset, and is influenced by old waterfall methods, but did provide a good foundation for the development[1]. After starting writing code based on the initial plan, refactoring proved an excellent way of adapting to changing requirements and new ideas.

A typical refactoring task starts with a new requirement or the need of a new function based on testing of a working prototype of the portal. The affected views, controller classes, and model classes/database tables are then refactored bottom-up. Coupled with unit testing, this is a fast and easy way of evolving and changing the system, and it proved to work well together with the other principles of work mentioned above.

8.3.4 Unit Testing

Unit testing is regarded as a mandatory in any agile software engineering. We have used JUnit (Section 5.4.5) to unit test our code. A unit test is simply a test case where one or more methods in the test object are run, and the output is compared to a correct predefined result. The unit testing was mainly applied at the controller layer, testing the methods used by the view layer without needing a finished user interface. In this way, the model layer classes were tested implicitly as the controllers would not work correctly without the right DAO support. Our unit testing has helped identify and correct many logical errors in the source code, and has saved a lot of debugging time as the unit tests run fast and easy within the IDE (Eclipse).

[1] "If it's working - Fine! If it's not working - Fix it!" - Jens Østergaard, Scrum coach

8.4 Challenges and Success Factors

Reuse repositories like OTS-Wiki have a tendency to not being reused by targeted developers due to under-critical relevance. Such repositories easily becomes "information graveyards" if the information does not have high relevance and value to the potential users. Another issue is potential heavy start-up costs related to initiate and populate a reuse repository.

In an article about populating software repositories, Poulin [Poulin, 1995] present IBM's experiences with a corporate Reusable Software Library (RSL). At the start the library is empty. It is hard to convince developers to even look at it because of this, just like an "library without books". Launching an intensive program to populate the repository leads to fast increasing of parts, but it is still hard to convince developers to use it because they know it mostly contains software of variable and even poor quality.

Further, Poulin [Poulin, 1995] states the following typical three-phase progression:

- The repository contains very *few parts*
- The repository contains many parts of *poor quality*
- The repository contains many parts of *little or no use*

A well populated repository also leads to many challenges. Detailed classification is needed to organize the large collection, but this could make the search confusing. In addition, it requires training to use it [Poulin, 1995].

Further Polin writes, to succeed it is important not to measure the value of the system by counting total lines of code, but instead focus on providing domain-specific content to ensure the usefulness of the RSL.

Polin's observations from the population of IBM's RSL are highly relevant for OTS-Wiki. To draw conclusions out of it, we must through OTS-Wiki try to offer:

Low start-up costs by incremental filling of the repository, and let search after not found components act as requirements for components not yet added to the repository to encourage and ease the process of adding new and relevant components.

Increased relevance for developers by providing references to actual users of the components with relevant knowledge (component responsible), lessons-learned, FAQs, component quality evaluations, integration cost, and development cases/scenarios etc.

A solution to both these issues could be a web based portal using wiki technology where the developers and users themselves controls and populates the repository. As suggested in [Ayala et al., 2007], this is done by using the open source collaboration principle. (See Social Computing in Section 3.5 Chapter 2) The challenge is to convince potential users about the potential in the OTS-Wiki, so that the portal gains a group of loyal and competently users to participate in the population process of the repository at an early stage.

8.5 Problems and Lessons Learned

8.5.1 Server Administration

The server applications, Subversion, Tomcat, and MySQL, have been hosted and administered by an external actor[2]. This has resulted in some delay and problems during the development of OTS-Wiki. The Tomcat web container has been especially problematic. We have run local installations of Tomcat to enable quick debugging, but when publishing the portal on the hosted server, some problems appeared. The main problem was related to refused connections by the local combination of Tomcat and the MySQL database on the server. Due to other tasks being prioritized above this problem, delays in the portal deployment appeared. We believe that the development and deployment of OTS-Wiki would have been easier and more reliable if the project had a dedicated server, free of other activity, to deploy the portal to. A server administrator within the development team is also preferable.

8.5.2 Limited Time

Time is a scarce resource in all software engineering projects. This implementation has been part of a Master thesis, where reporting work is widely important. Thus, the implementation work has been even more influenced by time limits. We have implemented the most important functionality, but have been forced to leave out many enhancing features.

An intial plan for the thesis was to use external actors as subjects for usability testing and evaluation. This was not possible due to the time issues. Thus, we have been forced to perform a simplified usability test internally. This has influenced the evaluation of the work, but hopefully the most important aspects still have been covered.

[2]The Cosiportal Project by Per Kristian Schanke

Part V

Conclusion and Further Work

Chapter 9

Conclusion

There exists a great variety of projects providing different functionality related to supporting the process of OTS component selection. So what differs OTS-Wiki from, e.g, SourceForge.net, Tigris.org, and Freshmeat?

One problem when selecting OTS software components is the the lack of structured documentation related to the selection and later the integration of the component. It would be very useful to have access to such information to reduce the cost of the selection process. It would also reduce the risk of using OTS components if the integration process already were documented.

The OTS-Wiki aims to give a solution to these problems. All resources related to a component is collected at one place. OTS-Wiki should provide references to actual users of the components, a component responsible with experience and knowledge about the component. Further, it should record and provide information about lessons-learned, FAQs, evaluation of component quality, integration cost, and development cases/scenarios etc. This is information of high value and relevance to the potential users of OTS-Wiki. This enables OTS-Wiki to act as a platform for structuring unstructured OTS knowledge found in other portals and web-sites.

Populating a repository such as OTS-Wiki is very expensive. This could mean high start-up costs. Another problem is the risk of not being reused. If the repository is populated by mostly general components with low relevance, it could end up as a "information graveyard"; Write once, never read.

To overcome these challenges, OTS-Wiki is using the wiki principle. OTS-Wiki will benefit from the social computing principles. The developers and users themselves controls and populates the repository, and the population is done in an incremental manner. This will probably lead to lower start-up cost and higher relevance for the users.

Chapter 10

Further Work

In this chapter we will outline and describe several suggestions on new functionality for OTS-Wiki. The time frame of this master thesis is limited, and the implementation we have performed only scratches the surface of the possible great potential which lies in OTS-Wiki concept.

10.1 Categorizing

The most important part of OTS-Wiki when it comes to usability is how easy it is for the user to find the desired information. Categorization of the components plays a essential role in the navigation efficiency. GOThIC (Section 3.2) is a method for building and maintaining an infrastructure of COTS components, and could be helpful to improve the usability of OTS-Wiki.

10.2 Selection Process Support

DesCOTS described in Section 3.3 is a system with several tools interacting to support the COTS software component selection process. The OTS-Wiki aims to be a web-based tool with much functionality provided by the DesCOTS system. DesCOTS functionality such as quality patterns support could be of great value to OTS-Wiki users when searching for suited components.

Such pattern-based search demands a certain structure on the evaluations so that the requirements could be compared and matched against the component evaluations to provide relevant search results.

10.3 User Profiling

Different users have different needs. By implementing functionality for customization of each user profile, each user could experience more efficient use of OTS-Wiki. Search results adjusted

to the user profile so that the relevance is higher is one way of taking advantage of extended profile functionality.

The user profile concept could be taken further by developing extensive logging functionality. The users habits when using OTS-Wiki could also influence on the information presented to the user. Data collected from users with similar profile settings and user habits are another source for user customized information. In its simplest form, it could be a list such as: "Users interested in this component were also browsing these components".

10.4 Automated Maintenance

A great challenge in Wiki-based portals like OTS-Wiki is to keep the material up-to-date. First of all it is dependent of the maintenance work to be done by OTS-Wiki users on a regular basis.

Some of this maintenance could possible be done without human interaction by the use if different types of web-crawlers. The Open Source Directory Ohloh[1] presented in Chapter 4, uses a web-crawler to monitor the development activity of the different projects listed in the directory.

10.5 Evaluation Support

The component evaluation functionality is one of the most important aspects of OTS-Wiki. The users will be able to find and share user experience about the different components listed in the wiki repository. The already implemented evaluation functionality are at this stage very basic. To increase the value and quality of the evaluation work done by the user, it could be very useful to develop extensive evaluation wizards to guide the user through the process. Not only could this have a good influence on the quality, but it could also lower the barriers for other users to contribute with their valuable experience and knowledge.

A evaluation wizard could contain detailed questions on different aspects concerning by example the use, installation or integration of the component. By using wizard based evaluation instead of free text evaluations, makes it easier to organize the experience data the user provides.

The wizards could also be customised to different component categories, as many evaluation criteria are not relevant to al types of components.

10.6 Change Management

The present implementation of OTS-Wiki does only log the date of the last change made to the component in the database. Of many reasons it would be useful to extend the change management capabilities. Logging the change history of a component is a possible improvement. Instead of overwriting the old version with a new one, both should be saved in the database along with all previous versions. This adds rollback possibilities so that changes may be undone if previous version is preferable. Storing this historical data may also become useful if this information of any reason is needed at a later stage.

Wikipedia[2] uses a version and revision control system. If changes made to an article are

[1] http://www.ohloh.net/
[2] http://en.wikipedia.org/wiki/Wikipedia:About

of poor quality or accidentally is changed, or someone tries to vandalise the content, it can easily be reversed back to it's original state. This is obviously functionality which should be implemented in future versions of OTS-Wiki.

10.7 User Levels

One of the main challenges in wiki based web portals is to ensure the quality of the information. There is always a risk of wrong or misleading information on the web, and the chance of errors is even bigger when many different individuals contributes. One way of controlling this is by introducing several levels of users.

The user hierarchy may look like this:

- User
- Trusted user
- Senior user
- Administrator

To contribute with any kind of information, one must create a user account and by this gain the status as an **User**. By contributing quality information over a certain period, the user status is upgraded to a **Trusted user**. Before the information provided by a regular **User** is published on OTS-Wiki, it should be approved by a **Trusted user**, **Senior user**, or **Administrator**. To become a **Trusted user**, a **Senior user** or higher has to grant this user level. The **Administrator** role is given to the persons operating OTS-Wiki.

10.8 Administration

The administration abilities of the implemented OTS-Wiki is limited to basic functionality. To ensure efficient administration of the wiki, there will be need for tools covering all aspects concerning the maintenance of OTS-Wiki. This could be extensive user administration including user level handling introduced in previous section. Another one is section administration allowing the administrators to easily add and remove content such as menus and menu items, and update the information in the different sections or add new sections.

10.9 Co-Evaluation

By co-evaluation we mean evaluating two or more components together. Related components could be evaluated together by comparing different evaluation criteria step-by-step, and pointing out strengths and weaknesses of each component. This would make the process of finding a suitable component easier when there are several similar alternatives.

10.10 Component Versioning

OTS components evolve continuously and new versions are released more or less regularly. OTS-Wiki needs to handle the different versions of the components in the database. A version

of a component may behave like a standalone component, but inherit some overall information common to all versions. Also, the linking of a component to other components or software may require certain versions of each component to work. This information should be kept by OTS-Wiki to ease the selection process.

10.11 Other Improvements

Other improvements and extra functionality:

- RSS-feed on news and newly added components
- Improve the presentation of component and component evaluations
- Customization of user-profile adding e.g. picture or different contact data

Extended user feedback possibilities:

- Mark components as deprecated
- Mark duplicated components or suggest merging when the same component appears two or more times
- Rate the evaluations after how useful the user finds them

Bibliography

[Abrahamsson et al., 2002] Abrahamsson, P., Salo, O., Ronkainen, J., and Warsta, J. (2002). Agile Software Development Methods: Review and Analysis.

[Ayala and Franch, 2006] Ayala, C. and Franch, X. (2006). A goal-oriented strategy for supporting commercial off-the-shelf components selection. *Lecture Notes in Computer Science : Reuse of Off-the-Shelf Components*, pages 1–15.

[Ayala et al., 2007] Ayala, C., Sørensen, C.-F., Franch, X., Conradi, R., and Li, J. (2007). Open Source Collaboration for Fostering Off-The-Shelf Components Selection. OSS, June 2007, Limerick, Ireland.

[Bass et al., 2003] Bass, L., Clements, P., and Kazman, R. (2003). *Software Architecture in Practice, Second Edition*. Addison-Wesley Professional.

[Beck, 1999] Beck, K. (1999). *eXtreme Programming Explained: Embrace Change*. The XP series. Addison-Wesley, Reading, Mass., US.

[CBSE, 2006] CBSE (2006). The 9th International SIGSOFT Symposium on Component-Based Software Engineering. Online: http://www.sei.cmu.edu/pacc/CBSE2006/, accessed: 2007-03-15.

[Charron et al., 2006] Charron, C., Favier, J., and Li, C. (2006). How Networks Erode Institutional Power, And What to Do About It. *Forrester Research: Social Computing*.

[Dean and Gravel, 2002] Dean, J. and Gravel, A., editors (2002). *COTS-Based Software Systems*, Orlando, Florida, USA. Springer-Verlag. LNCS 2255.

[Eide and Schanke, 2006] Eide, T. E. and Schanke, P. K. (2006). Going open: Guidelines for commercial actors to release software as open source. Depth Project, IDI, NTNU.

[Fielding, 1999] Fielding, R. T. (1999). Shared leadership in the apache project. *Commun. ACM*, 42(4):42–43.

[Franch et al., 2007] Franch, X., Grau, G., Quer, C., Lopez-Pelegrin, X., and Carvallo, J. P. (2007). Descots: Qm conceptual model. UPC and NTNU.

[Fraser et al., 2006] Fraser, S., Ågerfalk, P. J., Eckstein, J., Korson, T., and Rainsberger, J. (2006). Open Source Software in an Agile World. In *7th International Conference, XP 2006*, pages 217–220, Oulo, Finland. Springer. Panel at XP'2006. LNCS 4044.

[Geelan, 2006] Geelan, J. (2006). Social Computing: Oxymoron - or the Biggest New Thing Since The Web Itself? Blog: http://jeremy.linuxbloggers.com/social_computing_as_biggest_big_new_thing.htm, accessed: 2007-06-07.

[Gould and Lewis, 1985] Gould, J. D. and Lewis, C. (1985). Designing for usability: key principles and what designers think. *Commun. ACM*, 28(3):300–311.

[Grau et al., 2004] Grau, G., Quer, C., Carvallo, J. P., and Franch, X. (2004). Descots: a software system for selecting cots components. *Euromicro Conference, 2004. Proceedings. 30th*, pages 118–126.

[Grossman, 2005] Grossman, J. (2005). The 80/20 Rule for Web Application Security - Increase your security without touching the source code. *Web Application Security Consortium*.

[Heiss, 2007] Heiss, J. J. (2007). Writing Better Code: A Conversation With Sun Microsystems Technology Evangelist Brian Goetz. *Sun Developer Network*.

[Hemrajani, 2006] Hemrajani, A. (2006). *Agile Java Development With Spring, Hibernate and Eclipse*. Sams.

[Hinchcliffe, 2006] Hinchcliffe, D. (2006). Thinking Beyond Web 2.0: Social Computing and the Internet Singularity. Blog: `http://web2.socialcomputingmagazine.com/thinking\_beyond\_web\_20\_social\_computing\_and\_the\_internet\_sin.htm`, accessed: 2007-06-07.

[Hornbæk and Law, 2007] Hornbæk, K. and Law, E. L.-C. (2007). Meta-analysis of correlations among usability measures. In *CHI '07: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 617–626, New York, NY, USA. ACM Press.

[IBM, 2007] IBM (2007). Social Computing Group - Frequently Asked Questions. Online: http://www.research.ibm.com/SocialComputing/SCGFAQs.htm, accessed: 2007-06-07.

[Jaccheri and Torchiano, 2002] Jaccheri, L. and Torchiano, M. (2002). Classifying COTS products. In *European Conference on Software Quality*, Helsinki, Finland.

[Larman, 2001] Larman, C. (2001). *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and the Unified Process (2nd Edition)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

[Lebanidze, 2006] Lebanidze, E. (2006). Securing Enterprise Web Applications at the Source: An Application Security Perspective. *OWASP - The Open Web Application Security Project*.

[Li et al., 2004] Li, J., Bjørnson, F. O., Conradi, R., and Kampenes, V. B. (2004). An empirical study of variations in cots-based software development processes in norwegian it industry. In *METRICS '04: 10th International Symposium on Software Metrics*, pages 72–83, Washington, DC, USA. IEEE Computer Society.

[Li et al., 2005] Li, J., Conradi, R., Slyngstad, O. P. N., Torchiano, M., Morisio, M., and Bunse, C. (2005). Preliminary Results from a State-of-Practice Survey on Risk Management in Off-The-Shelf Component-Based Development. In Franch, X. and Port, D., editors, *4th Int. Conf. on COTS-based Software System (ICCBSS'05)*, pages 278–288, Bilbao, Spain. Springer-Verlag. LNCS 3412.

[Morisio and Torchiano, 2002] Morisio, M. and Torchiano, M. (2002). Definition and Classification of COTS: a proposal. In *[Dean and Gravel, 2002]*, pages 165–175.

[Netcraft, 2007] Netcraft (2007). May 2007 Web Server Survey. Online: http://news.netcraft.com/archives/web_server_survey.html, accessed: 2007-06-07.

[NTNU, 2007] NTNU (2007). Community Collaboration and Web-Intelligence to Support Development with Off-The-Shelf Software Components. Small or medium-scale focused research (STREP) proposal.

[Ochs et al., 2001] Ochs, M., Pfahl, D., and Chrobok-Diening, G. (2001). A Method for Efficient Measurement-based COTS Assessment and Selection - Method Description and Evaluation Results. In *IEEE 7th Int. Software Metrics Symposium*, pages 285–296, London, UK. IEEE.

[Poulin, 1995] Poulin, J. S. (1995). Populating software repositories: incentives and domain-specific software. *J. Syst. Softw.*, 30(3):187–199.

[Raymond, 2001] Raymond, E. S. (2001). *The Cathedral & the Bazaar (paperback)*. O'Reilly.

[Richardson, 2006] Richardson, C. (2006). *POJOs in Action - Developing Enterprise Applications with Lightweight Frameworks*. Manning Publications inc.

[Sommerseth, 2006] Sommerseth, M. (2006). Component based system development in the norwegian software industry. Master's thesis, IDI, NTNU.

[Sørensen, 2002] Sørensen, C.-F. (2002). Extreme Programming - A Brief Introduction. http://www.idi.ntnu.no/ carlfrs/.

[Torchiano et al., 2002] Torchiano, M., Jaccheri, L., Sørensen, C.-F., and Wang, A. I. (2002). COTS Products Characterization. In *14th International Conference on Software Engineering and Knowledge Engineering (SEKE'02)*, pages 335–338, Ischia, Italy. ACM Press.

[Torchiano and Morisio, 2004] Torchiano, M. and Morisio, M. (2004). Overlooked aspects of cots-based development. *IEEE Softw.*, 21(2):88–93.

[Voas, 1998] Voas, J. (March 1998). OTS: The Economical Choice? *IEEE Software*, 15(2):16–19.

[von Hippel and von Krogh, 2003] von Hippel, E. and von Krogh, G. (2003). Open source software and the "private-collective" innovation model: Issues for organization science. *Organization Science*, 14(2):209–223.

[Warsta and Abrahamsson, 2003] Warsta, J. and Abrahamsson, P. (2003). Is open source software development essentially an agile method? In *Proceedings of the 3rd Workshop on Open Source Software Engineering, 25th International Conference on Software Engineering*, pages 143–147, Portland, Oregon.

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

[Wikipedia, 2007] Wikipedia (2007). Wikipedia - The Free Encyclopedia. Online: http://www.wikipedia.org/, accessed: 2007-03-05.

[Øyvind Hauge and Røsdal, 2006] Øyvind Hauge and Røsdal, A. (2006). A survey of industrial involvement in open source. Master's Thesis/Depth Project, IDI, NTNU.

Part VI

Appendices

Appendix A

Use Case Specification

A.1 User Actions

Figure A.1 depicts the use cases related to the User actor. Before a user adopts a role as a Provider, an Evaluator, or a Selector, these actions are valid. The use cases are described below.

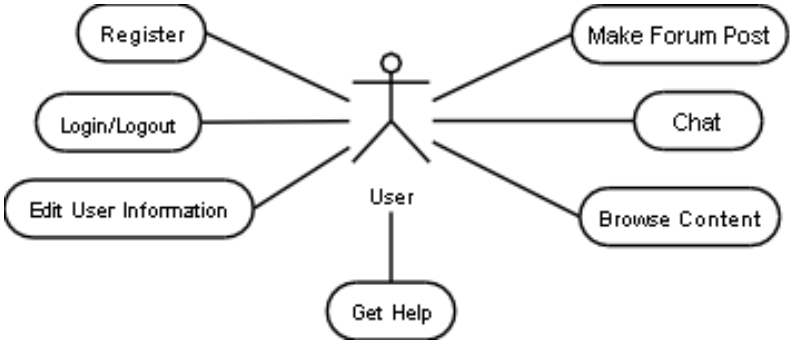

Figure A.1: User Actions

UC1.1 Register

To become a member of the community, the user has to register. A registered user may become a Provider, an Evaluator, or a Selector, depending on the tasks performed. The system administrator assigns different roles to the users enabling multiple levels of access rights. Table A.1 describes the use case.

Table A.1: UC1.1 Register

Name: UC1.1 Register	**Priority:** High
Intent: Register as user of the system	**Actor:** User
Precondition: None	
Main Success Scenario:	

1. The actor access the registration page
2. The actor provides a user profile (username, password, and e-mail mandatory)
3. The actor chooses relevant interests from a list
4. The actor submits the information
5. The system validates the input
6. The system stores the actor in the database
7. The system sends an e-mail to the user confirming the registration
8. The actor can now login on the system

Alternative Scenarios:

A The chosen username exist

1 The actor is asked to provide a different username

B The actor has not provided the needed input

1 The actor is asked to provide the missing input

UC1.2 Login/Logout

Registered users may log in and out of the system. Table A.2 describes the use case.

Table A.2: UC1.2 Login/Logout

Name: UC1.2 Login/Logout	**Priority:** High
Intent: Log in and out users	**Actor:** User
Precondition: The actor is registered as a user	
Main Success Scenario:	

1. The actor enters username and password in the login dialog
2. The system looks up the user in the database
3. The system logs in the actor
4. The actor is logged in until the session is closed or the actor manually logs out

Alternative Scenarios:

A Wrong username or password provided

1 The actor may retype the username and password or follow the next steps:

2 The actor provides an e-mail address

3 If a user with the provided e-mail address exist, the system sends a reminder to the user with the username and password

UC1.3 Edit User Information

This feature enables the registered users to maintain the user information. Table A.3 describes the use case.

Table A.3: UC1.3 Edit User Information

Name: UC1.3 Edit User Information	**Priority:** Medium
Intent: Maintain updated user information	**Actor:** User
Precondition: The actor is logged in	
Main Success Scenario:	

1. The actor accesses the user profile
2. The actor updates the information and submits the changes
3. The system validates the input
4. The system stores the changes in the database

Alternative Scenarios:

A The actor has not provided the needed input

C1.1 The actor is asked to provide the missing input

UC1.4 Make Forum Post

A public forum is an important feature in a web community. Registered users may post new topics or replies related to specific components or general discussions. Unregistered users have read-only access to the forum. Table A.4 describes the use case.

Table A.4: UC1.4 Make Forum Post

Name: UC1.4 Make Forum Post	**Priority:** Medium
Intent: Community and component forums	**Actor:** User
Precondition: The actor is logged in	
Main Success Scenario:	

1. The actor may start a new topic
2. The actor may post a reply to existing topics
3. The actor may edit her own posts
4. The actor may delete her own posts
5. The system offers a textual search to find wanted topics and posts

Alternative Scenarios:

A Read forum content

1 The actor may read forum content without being logged in

UC1.5 Chat

Registered users may chat with other users currently online. This feature enables quick flow of information and helps building the community. Table A.5 describes the use case.

Table A.5: UC1.5 Chat

Name: UC1.5 Chat	**Priority:** Low
Intent: Live chat rooms for online users	**Actor:** User
Precondition: The actor is logged in	
Main Success Scenario:	

1. The actor access the chat lobby
2. The actor choses the wanted chat room (sorted by topic)
3. The actor may chat with the other users in the chat room

Alternative Scenarios:

A No chat activity

1 The chat rooms have no participants

2 The chat room is marked as activated when two or more users are using it

UC1.6 Browse Content

This feature covers access to all information that is available read-only to all users, registered or unregistered. As an open community, most of the content is commonly available read-only. All links, resources, news, events, components, evaluations, ratings, documentations, and forum posts are included. Table A.6 describes the use case.

Table A.6: UC1.6 Browse Content

Name: UC1.6 Browse Content	**Priority:** High
Intent: Browse read-only content	**Actor:** User
Precondition: None	
Main Success Scenario:	

1. All content is available read-only to unregistered users
2. The actor can browse the community, accessing links, resources, news, events, and documentations
3. The actor can download the system's source code

Alternative Scenarios:

A The actor wants to contribute to the community

1 The actor becomes a Provider (Section A.2)

B The actor wants to evaluate a component

1 The actor becomes an Evaluator (Section A.3)

C The actor wants to seek specific components

1 The actor becomes a Selector (Section A.4)

UC1.7 Get Help

This feature provides help and information about the functions and features on each page in the system. The actor has to be logged in to access the help feature. Table A.7 describes the use case.

Table A.7: UC1.7 Get Help

Name: UC1.7 Get Help	**Priority:** Low
Intent: Help to the functions and pages in the system	**Actor:** User
Precondition: The actor is logged in	
Main Success Scenario:	

1. The actor accesses the help function on a page
2. The system presents a help dialog with descriptions of the functions and features on the page
3. The actor may contact system administrator for further help

A.2 Provider Actions

Figure A.2 depicts the use cases related to the Provider actor. The use cases are described below.

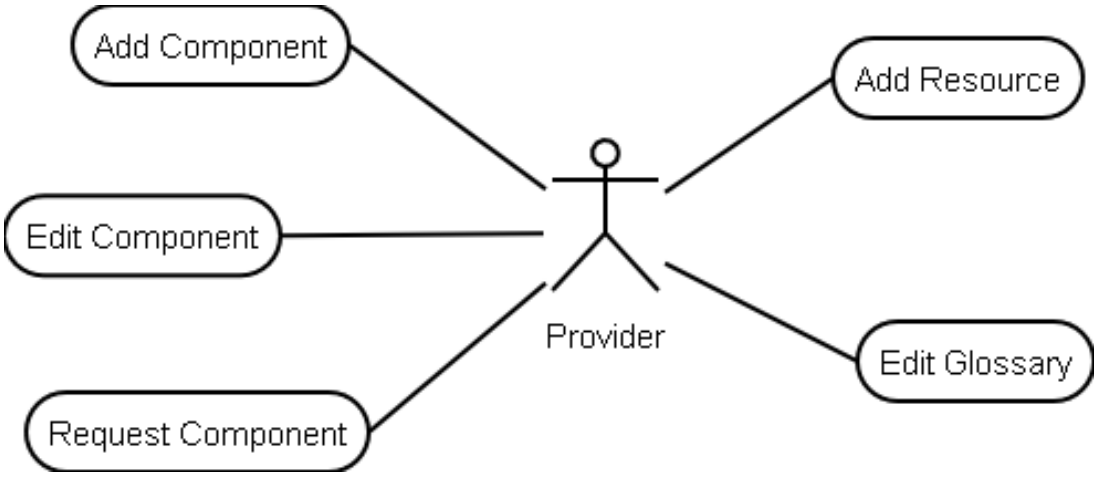

Figure A.2: Provider Actions

UC2.1 Add Component

Adding new components is a crucial task in OTS-Wiki. Without a widely populated database, the portal has little or no value to the users. The Provider actor adds new components, including the information needed to find and start using the components. All component information is available read-only to unregistered users. Table A.8 describes the use case.

Table A.8: UC2.1 Add Component

Name: UC2.1 Add Component	**Priority:** High
Intent: Add new components to the community	**Actor:** Provider
Precondition: The actor is logged in	
Main Success Scenario:	

1. The actor accesses the 'Add Component' link from the menu
2. The actor provides the information about the component
3. The actor submits the new component
4. The system validates the input
5. The system stores the component in the database

Alternative Scenarios:

A The actor has not provided the needed input

1 The actor is asked to provide the missing input

B The component already exists

1 The actor may look at or edit the existing component

2 The actor may start registering another component

UC2.2 Edit Component

One of the main ideas of a Wiki-inspired system is that the content is user-editable. This feature will enable the Provider actor to edit existing components, adding new versions, or changing facts and information. Table A.9 describes the use case.

Table A.9: UC2.2 Edit Component

Name: UC2.2 Edit Component	**Priority:** High
Intent: Edit an existing component	**Actor:** Provider
Preconditions: 1. The actor is logged in 2. The component exists 3. The actor has access to edit the component	
Main Success Scenario:	

1. The actor accesses the component to edit
2. The actor accesses the 'Edit' link
3. The actor edits the component information
4. The actor submits the edited component information
5. The system validates the input
6. The system stores the component in the database
7. The system maintains a log of the component's history

Alternative Scenarios:

A The actor has not provided the needed input

1 The actor is asked to provide the missing input

UC2.3 Request Component

We identify a constant need for new components. Many stakeholders may have ideas that are not yet implemented as OTS components. This feature will enable the Provider actor to request new components by providing wanted categories, requirements, and technologies. Table A.10 describes the use case.

Table A.10: UC2.3 Request Component

Name: UC2.3 Request Component	**Priority:** High
Intent: Request new components	**Actor:** Provider
Precondition: The actor is logged in	
Main Success Scenario:	

1. The actor accesses the 'Request Component' link from the menu
2. The actor chooses the wanted categories
3. The actor chooses the wanted requirements
4. The actor chooses the wanted technologies
5. The actor provides a textual description of the wanted component
6. The system validates the input
7. The system performs a search for candidate components in the database
8. The system stores the request in the database

Alternative Scenarios:

A Candidate components already exist in the database

1 The system presents the candidate components

2 The actor is asked if the candidate components provide adequate coverage

3 The actor either cancels or submits the request

UC2.4 Add Resource

The system maintains a list of additional resources not related to specific components, but that are relevant to component based development or other software engineering topics. A user acting as a Provider may add resources to the this list. The list is available read-only to unregistered users. Table A.11 describes the use case.

Table A.11: UC2.4 Add Resource

Name: UC2.4 Add Resource	**Priority:** Medium
Intent: Add external resources	**Actor:** Provider
Precondition: The actor is logged in	
Main Success Scenario:	

1. The actor accesses the 'External Resources' page from the menu
2. The actor accesses the 'Add Resource' link
3. The actor provides the name, hyperlink , and a description of the resource
4. The system validates the input
5. The system stores the resource in the database

Alternative Scenarios:

A The resource with the given name already exists

1 The system compares the information of the existing and the new resource

2 If the information is identical, the new resource is ignored

3 If the information is different, the actor is asked to merge the information

UC2.5 Edit Glossary

The glossary is an open feature that a user acting as a Provider can edit. It contains descriptions to terms and abbreviations that is relevant to the community in particular, and computer engineering in general. Unregistered users have read-only access to the glossary. Table A.12 describes the use case.

Table A.12: UC2.5 Edit Glossary

Name: UC2.5 Edit Glossary	**Priority:** Medium
Intent: Maintain a glossary of relevant terms	**Actor:** Provider
Precondition: The actor is logged in	
Main Success Scenario:	

1. The actor accesses the 'Glossary' link from the menu
2. The actor can edit existing items or add a new item
3. The actor provides a name and a description
4. The actor submits the new or edited item
5. The system validates the input
6. The system stores the glossary item in the database

Alternative Scenarios:

A The actor has not provided the needed input

1 The actor is asked to provide the missing input

A.3 Evaluator Actions

Figure A.3 depicts the use cases related to the Evaluator actor. The use cases are described below.

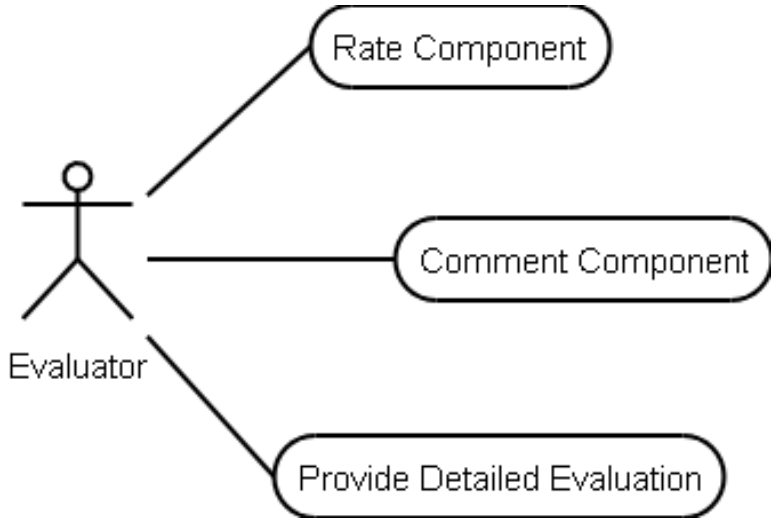

Figure A.3: Evaluator Actions

UC3.1 Rate Component

Rating is the lowest level of component evaluation. The actor rates the general impression of the component on a scale 1-6. The actor has to be logged in to rate a component, but the ratings are anonymous. The highest rated components are presented on the portal's main page for quick reference.

Table A.13: UC3.1 Rate Component

Name: UC3.1 Rate Component	**Priority:** High
Intent: Rating - The lowest level of evaluation	**Actor:** Evaluator
Precondition: The actor is logged in	
Main Success Scenario:	

1. The actor accesses the component to evaluate
2. The actor accesses the 'Rate Component' link
3. The actor rates the component on a scale of 1-6
4. The system stores the rating in the database

UC3.2 Comment Component

Commenting a component is the middle level of evaluation. The comment is a general review of the component, not related to a specific quality or feature. The actor is encouraged to write some words about his or hers general impression of the portal that may be of interest to other users.

Table A.14: UC3.2 Comment Component

Name: UC3.2 Comment Component	**Priority:** High
Intent: Commenting - The middle level of evaluation	**Actor:** Evaluator
Precondition: The actor is logged in	
Main Success Scenario:	

1. The actor accesses the component to evaluate
2. The actor accesses the 'Comment Component' link
3. The actor provides a textual comment to describe the subjective view of the component
4. The system validates the input
5. The system stores the comment in the database

Alternative Scenarios:

A The actor has not provided the needed input

1 The actor is asked to provide the missing input

UC3.3 Provide Detailed Evaluation

The detailed evaluation includes both a rating and a comment aimed at specific concerns, qualities, or features in the component. This makes possible a more comprehensive and detailed evaluation.

Table A.15: UC3.3 Provide Detailed Evaluation

Name: UC3.3 Provide Detailed Evaluation	**Priority:** High
Intent: Detailed evaluation - The highest level of evaluation	**Actor:** Evaluator
Precondition: The actor is logged in	
Main Success Scenario:	

1. The actor accesses the component to evaluate
2. The actor accesses the 'Evaluate Component' link
3. The actor provides a rating and/or a comment on the following criterias:
   - Installation
   - Documentation
   - Integration
   - Degree of goal achievement
4. The system validates the input
5. The system stores the evaluation in the database

Alternative Scenarios:

A The actor has not provided the needed input

1 The actor is asked to provide the missing input

A.4 Selector Actions

Figure A.4 depicts the use cases related to the Selector actor. This includes means to find components and information stored in the portal. The use cases are described below.

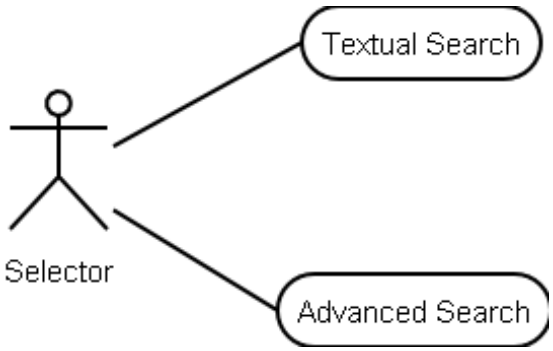

Figure A.4: Selector Actions

UC4.1 Textual Search

A textual search finds components based on a search string. The search may look in all text fields in the database, or use some sort of indexing framework or algoritm.

Table A.16: UC4.1 Textual Search

Name: UC4.1 Textual Search	**Priority:** High
Intent: Provide a textual search for components	**Actor:** Selector
Precondition: None	
Main Success Scenario:	

1. The actor provides a search string
2. The actor presses the search button
3. The system performs a textual search for components using the name and description
4. The system presents the matching components for the user to access

Alternative Scenarios:

A No matching components found

1 The actor is informed and asked to provide a different search string

UC4.2 Advanced Search

The advanced search performs a search based on given categories, technologies, platforms, or tags, enabling the actor to define some basic requirements to find relevant components. An additional search string may be included to narrow the search results.

Table A.17: UC4.2 Advanced Search

Name: UC4.2 Advanced Search	**Priority:** High
Intent: Provide a more detailed search for components	**Actor:** Selector
Precondition: None	
Main Success Scenario:	

1. The actor accesses the 'Advanced Search' link
2. The actor picks the wanted criteria to base the search on (category, technology, platform, or tag)
3. The actor pick one or more element from the list
4. The actor provides an additional search string (optional)
5. The system performs a search for components based on the chosen categories, technologies, platforms, or tags. The additional search string is used when applicable
6. The system presents the matching components for the user to access

Alternative Scenarios:

A No matching components found

1 The actor is informed and asked to provide different search criterias

A.5 Administrator Actions

Figure A.5 depicts the use cases related to the Administrator actor. The use cases are described below.

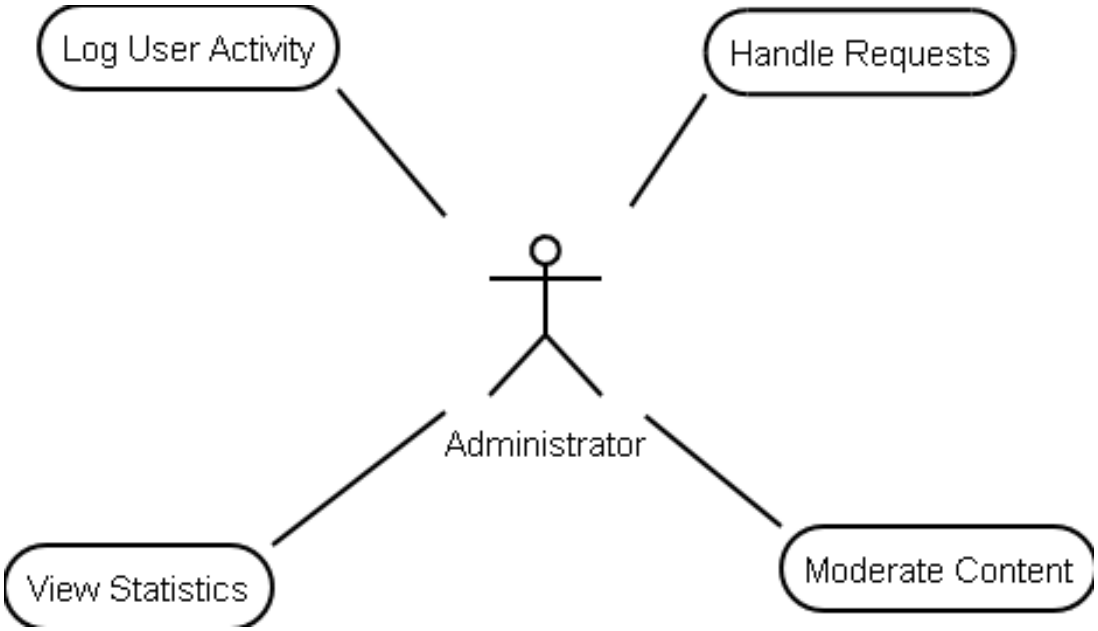

Figure A.5: Administrator Actions

UC5.1 Log User Activity

Logging user activity can be done automatically by the portal. The task of the administrator can be to customize and tune the logging to detect certain activities or trends.

Table A.18: UC5.1 Log User Activity

Name: UC5.1 Log User Activity	**Priority:** Medium
Intent: Log and monitor user activity to enhance the portal	**Actor:** Administrator
Precondition: 1. The actor is logged in 2. The actor is an administrator	
Main Success Scenario:	

1. The system logs user activity (accessed pages and components)
2. The actor accesses the 'User Activity' link
3. The system presents the user activity based on the following criterias:
   - Single user activities
   - Grouped user activities
   - Popular components
   - Popular searches

UC5.2 Handle Requests

Requests for new components and features must be handled by an administrator. A “case” may be opened for each request to track the progress and status.

Table A.19: UC5.2 Handle Requests

Name: UC5.2 Handle Requests	**Priority:** Low
Intent: Handle requests for new components and functionality	**Actor:** Administrator
Precondition: 1. The actor is logged in 2. The actor is an administrator	
Main Success Scenario:	

1. The actor accesses the ’Handle Requests’ link
2. The system presents a list of unhandled requests from the users
3. The actor accesses a request and performs the needed task
4. The system marks the request as handled

UC5.3 View Statistics

Using portal statistics, needs for further improvements and changes mat be identified. This is an important administrator task.

Table A.20: UC5.3 View Statistics

Name: UC5.3 View Statistics	**Priority:** Low
Intent: View general portal statistics	**Actor:** Administrator
Precondition: 1. The actor is logged in 2. The actor is an administrator	
Main Success Scenario:	

1. The actor accesses the 'View Statistics' link
2. The system presents the statistics page containing data, such as:
   - Portal statistics - number of hits
   - User statistics - counts and contributions
   - Component statistics - number of hits, ratings, comments, and evaluations

UC5.4 Moderate Content

An administrator has write access to all the data in the portal, enabling moderation of incorrect, broken, or illegal content. Such activities may increase the quality or the stored information.

Table A.21: UC5.4 Moderate Content

Name: UC5.4 Moderate Content	**Priority:** Medium
Intent: Moderation of the portal content	**Actor:** Administrator
Precondition: 1. The actor is logged in 2. The actor is an administrator	
Main Success Scenario:	

1. The actor has access to edit all the content in the portal
2. The actor may perform the following tasks:
   - Change the component information (The administrator activity is not recorded in the component's history)
   - Delete users, components, comments, evaluations, glossary items, news, and resources
   - Moderate forums and chat rooms for unwanted content

Appendix B

OTS-Wiki Screenshots

Here, we present screenshots of some of the most important functionality of the implemented version of OTS-Wiki.

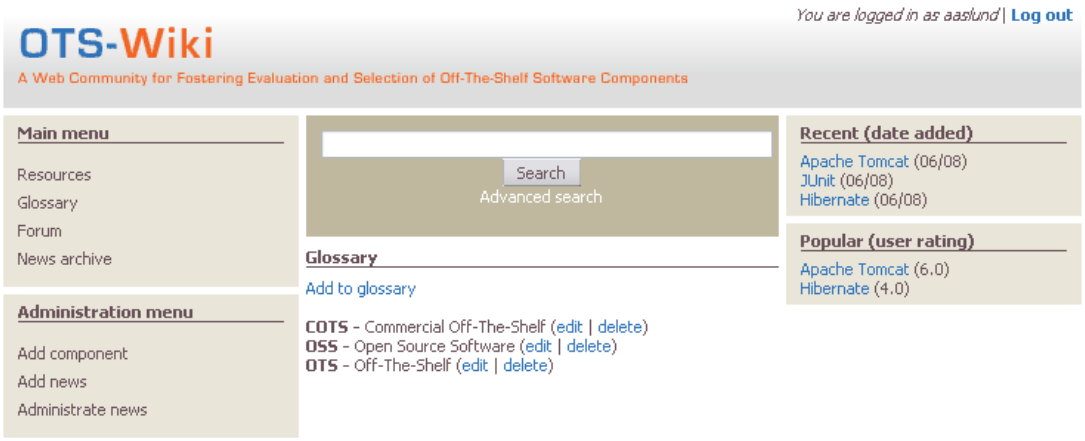

Figure B.1: Editing the glossary

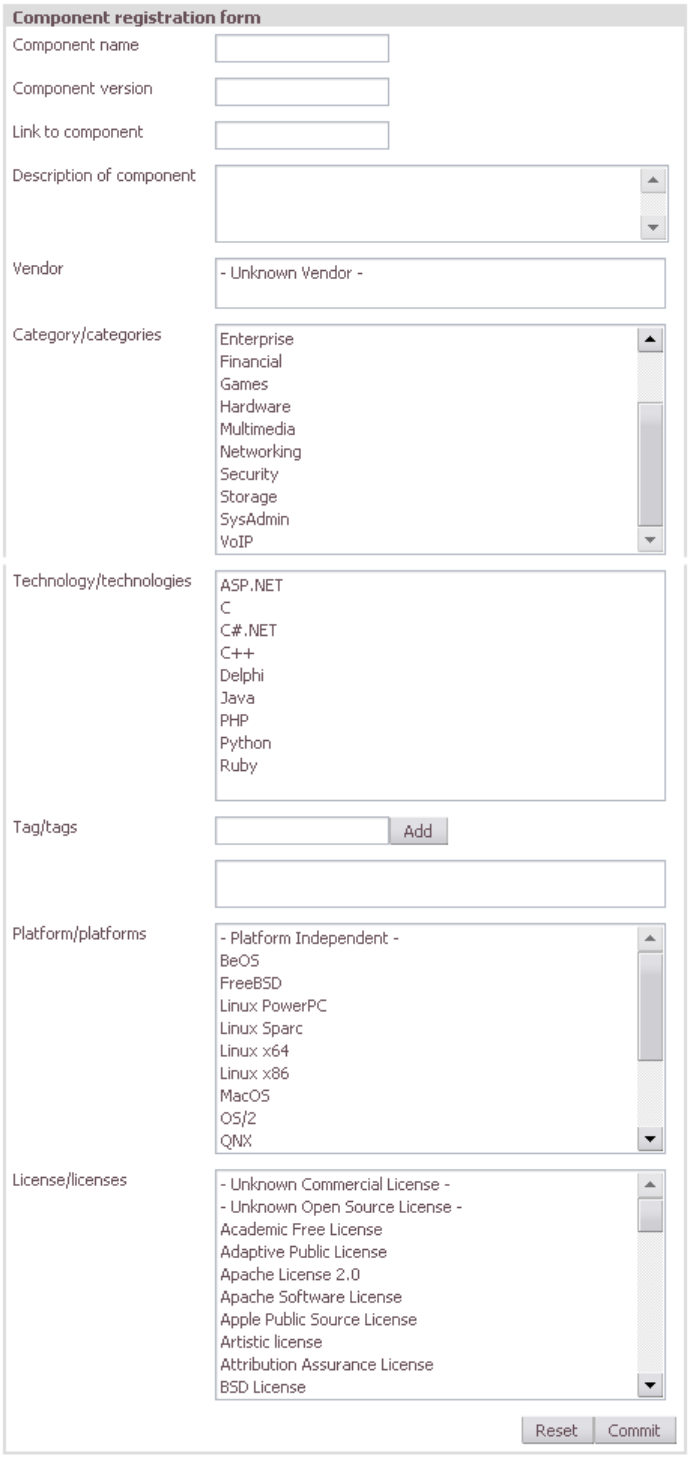

Figure B.2: Adding a new component

Figure B.3: Viewing a component

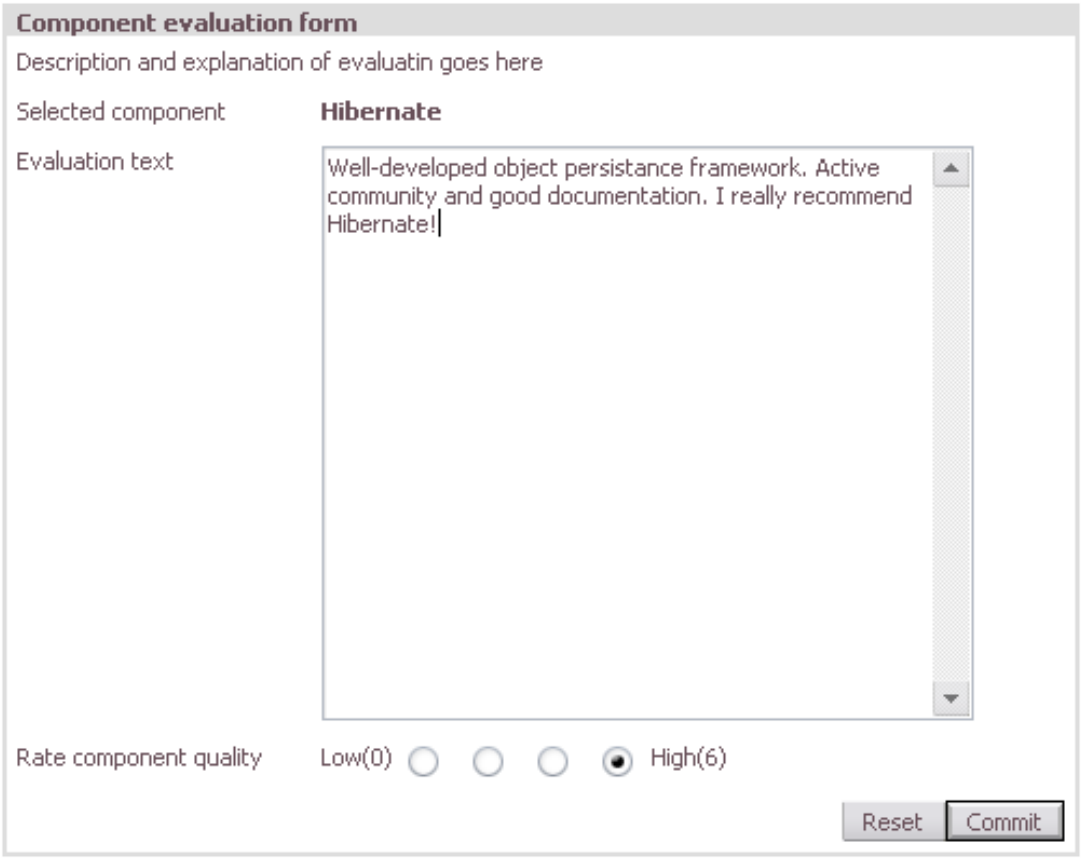

Figure B.4: Rating and commenting a component

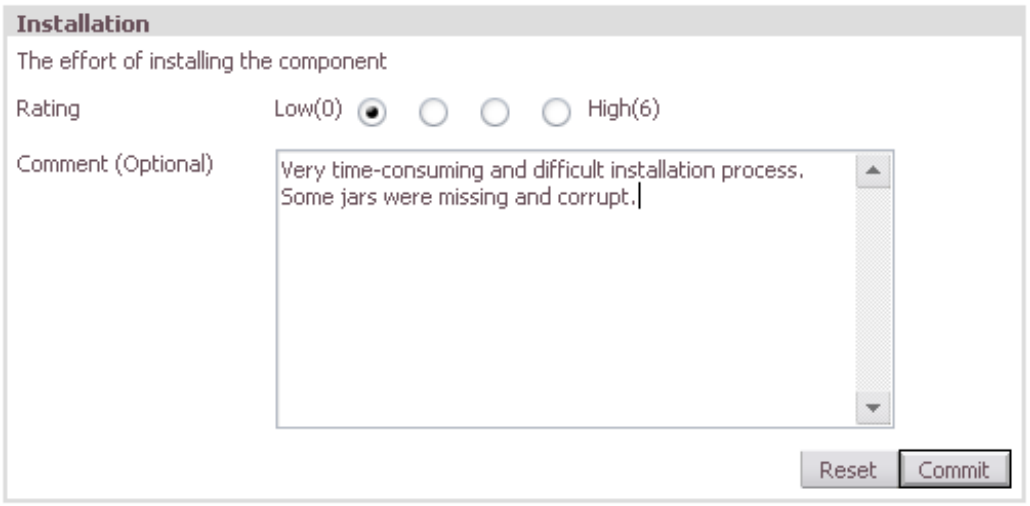

Figure B.5: Evaluating the installation process

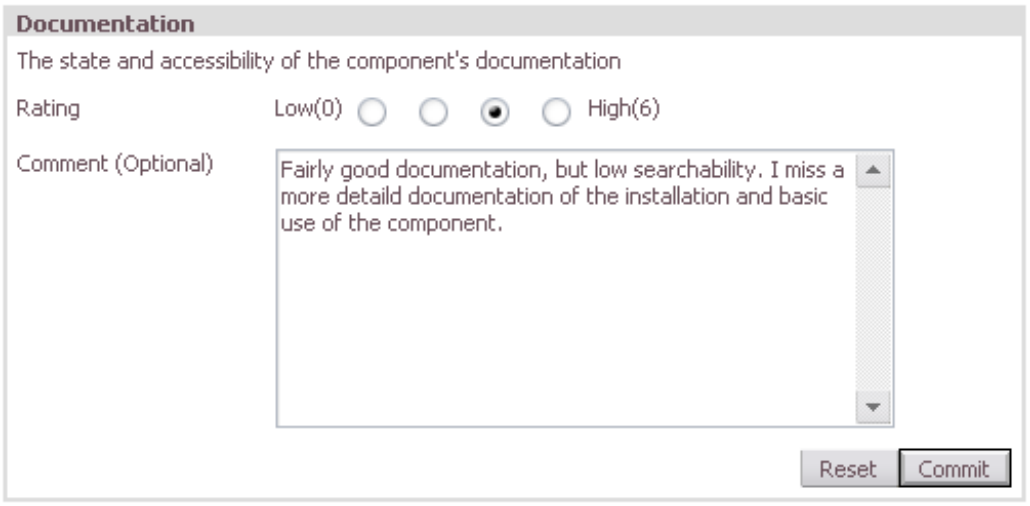

Figure B.6: Evaluating the component's documentation

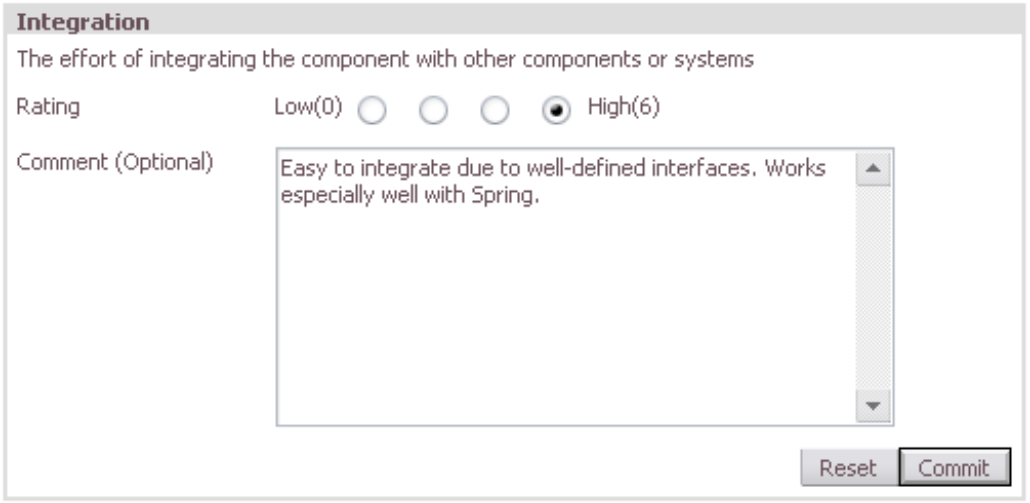

Figure B.7: Evaluating the integration process

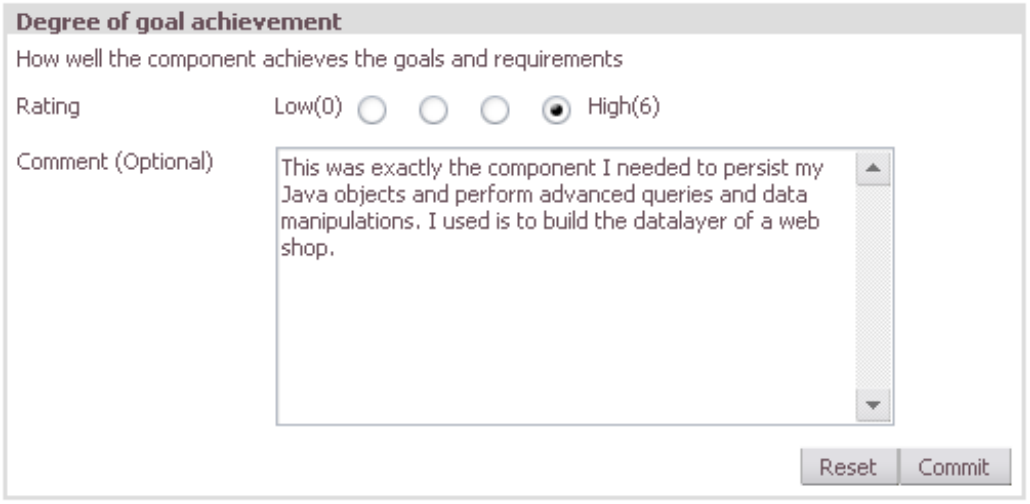

Figure B.8: Evaluating the degree of goal achievement

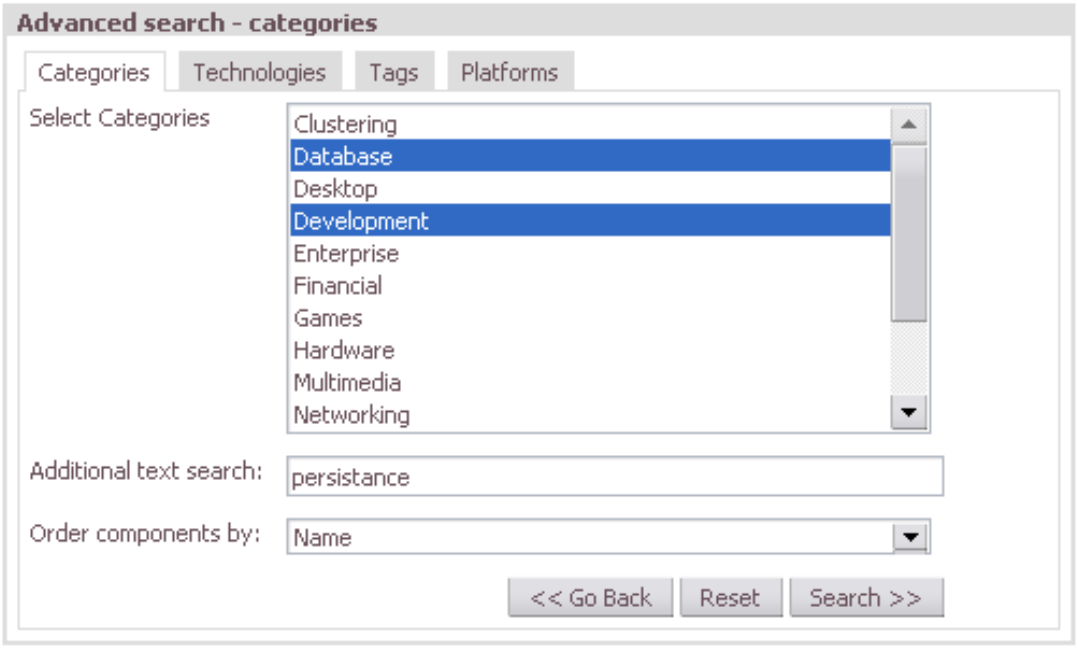

Figure B.9: The advanced search

Figure B.10: Search results