University of Groningen
Faculty of Mathematics and Natural Sciences
Department of Mathematics and Computing Science
Software Engineering and Architecture group

Requirements Specification

Open Pattern Repository

<http://openpatternrepository.googlecode.com>

-DRAFT-
v 1.0r
Groningen, October 7, 2009

Authors

Name	Acronym	E-Mail
Manteuffel, Christian	cm	c.manteuffel@student.fontys.nl
Verspai, Martin	mv	m.verspai@student.fontys.nl

Revision History

Version	Author	Date	Description
0.1	cm	09/08/09	Set up template conform to IEEE/ANSI 830-1998
0.2	cm	09/09/09	Added Purpose and Scope
0.2r	mv	09/10/09	Reviewed document and corrected typos
0.3	cm	09/11/09	Components, HLUC, Quality Attributes and supplementary requirements
0.4	cm,mv	09/15/09	Specific use cases
0.4r	cm	09/16/09	Review first part and HLUC
0.5	mv	09/17/09	Added and refined SPUCs
0.6r	cm	09/17/09	Added outlines and reviewed
0.7	cm	09/22/09	Added customer notes (refer to 10.../20.../correspondence_21_09.txt) HLUC(2, 6, 9) SPUC(2.2, 2.3, 4.3), Glos category, A3
0.8	cm,mv	09/23/09	Updating Glossary(OpenID, artifact, force, pattern, repository, template (removed: category, context, description, relationship, solution, source)), 1.2 Scope, 2.1 Stakeholder, 2.2 Product Perspective, 2.3 User characteristics , HLUC(1, 5, 6,7, 9, 10, 11, 12), SPUC(1.1-2, 2.1, 4.1, 4.3, 5.1, 10.1-3, 11.1-2, 12.1-3), traceability matrix SUPR(I1, I2, Q10, Q11, Q13, Q20), general constraints
0.8r	cm,mv	09/24/09	Review
0.9	cm,mv	09/25/09	Typos, changes: 1.2 Scope, 2.1 Stakeholders, 2.2 Product perspective, 2.3 User characteristics, HLUC(2, 6, 7, 8, 9, 10, 11), SUPR(I1,I2,Q13) SPUC(1.1, 1.2, 2.1, 4.2, 4.3, 6.1, 6.2, 6.3, 6.4, 9.1, 9.3, 10.1, 11.1)
1.0	mv	10/05/09	Corrected typos, changed glossary(pattern, template), user rights graphics, HLUC(1, 5, 6), SPUC(2.2, 4.1, 6.3, 11.1)
1.0r	mv	10/06/09	Reviewed introduction, Overall description

Legend: r = reviewed

Contents

List of Figures

List of Tables

Glossary

API
: =Application Programming Interface; is an interface in computer science that defines the ways by which an application program may request services from libraries and/or operating systems.

artifact
: An artifact represents a document or resource that is used in a software project. For instance design diagrams, source code or reports but also intangible resources like revisions within a versioning system.

CASE
: Computer-Aided Software Engineering; automated tools that can be used in the software development process.

driver
: Drivers are the reasons why a pattern has been chosen. Forces, relationships or consequences can be referenced as drivers.

force
: A force denotes any aspect of the problem that should be considered when solving it, such as requirements the solution must fulfill, constraints that must be considered and desirable properties the solution should have.[1]

OpenID
: OpenID is an open, decentralized standard for authenticating users which can be used for access control, allowing users to log on to different services with the same digital identity where these services trust the authentication body. OpenID replaces the common login process that uses a login-name and a password, by allowing a user to log in once and gain access to the resources of multiple software systems.

OPR
: =Open Pattern Repository

pattern
: In this document the term pattern refers to patterns and technologies. A pattern is a reusable solution to a common problem that cannot be transformed into code directly. It is a reusable template for solving a problem. The repository focuses on software pattern but allows any other kind of pattern too.
A technology like a framework, middleware or API can be described in an equal way compared to patterns [2].
At least both descriptions define a context, a problem and a solution.

repository
: Within a repository the patterns are stored and maintained.

RUG
: =University of Groningen

SEARCH
: =Software Engineering and Architecture group

tag
: Tags are keywords which are used to decorate a pattern with attributes similar to the technology that is used in blog software. Tags are searchable and can limit search results. Unlike categories, tags are not bound to any classification and don't have to be related to patterns either. Tags could also group metadata about the author or available artifacts like UML-diagram or Java-implementation. In conclusion tags are not bound to any classifications or domains.

template
: A template is a predefined structure of a pattern description that defines how the pattern will be captured. For instance, if the problem will be described textural or if it is subdivided by a list of defined forces.
Depending on the structure of the template, the input mask of the user interface will be influenced.

wizard
: A wizard is a user interface element, which leads the user through several steps in a specific sequence, to perform a task. Sometimes it may be easier to perform tasks using a wizard, especially for complex or infrequently performed tasks where the user is unfamiliar with the steps involved. Within the pattern repository a wizard is used to simplify the process of adding a pattern.

1 Introduction

1.1 Purpose

This document describes the requirements of the Open Pattern Repository (OPR), a publicly available online repository for patterns and technologies (hereinafter:patterns). Although the OPR provides additional functionality for software patterns, the repository can be used for any kinds of patterns like analysis patterns or organizational patterns.

The OPR is a project of the Software Engineering and Architecture Group (SEARCH) at the University of Groningen (RUG).

This document addresses members of the pattern community, software engineering practitioners and system developers.

It is structured and written according to the Recommended Practice for Software Requirements Specifications described in IEEE 830-1998[3].

1.2 Scope

The Open Pattern Repository will provide a **publicly available online repository to store, browse, search, update and delete patterns**. It shall provide a user web interface that can be used to easily **contribute to the pattern repository**, **explore and search patterns along different aspects**, but also to **document design decisions and generate project specific documentation**. Another interface will be an online **API** which allows CASE-tools to integrate parts of the OPR functionality. The OPR project members attach great importance to involve the pattern community in all phases of the project.

Three main parts can be identified:

- A component to manage patterns
- A tool to capture, document and explore pattern-related design decisions
- A component to configure and administer the OPR.

1.3 Definitions, acronyms and abbreviations

Please refer to the glossary for a list of definitions, acronyms and abbreviations.

1.4 References

Please refer to the bibliography for a list of all documents referenced in this article.

1.5 Overview

Section 2 of this document provides a use case model. It does not state specific requirements. Instead it describes the known actors and their interactions with the OPR from a high level view. It also lists factors and assumptions, that affect the requirements stated in this document. Section 3 describes the proper user requirements. Section 3.1 describes use case reports as major requirement specification artifacts. Section 3.2 captures requirements that are not included in the use cases.

2 Overall Description

The purpose of this section is to provide a brief description of the project and to describe known actors and their interaction with the system. The section presents a high level view. For more detailed information, please see the individual use case reports in the specific requirements.

2.1 Main Stakeholders of the OPR

Pattern Community Uses and shapes the information within the system by adding, changing, exploring and reviewing pattern descriptions. Furthermore, members of the pattern community give feedback and might actively be involved in the development.

Software Architects Software architects use the repository to document their project related design decisions and as a source of pattern knowledge to support them during the design process.

Developers Internship students are developing the OPR system. Developers include architects, testers and quality engineers.

Administrators Must integrate the system into a network infrastructure. Their concern is, that the system has an easy setup and causes no problems with existing systems.

2.2 Product perspective

Although several online repositories for patterns exist, the Open Pattern Repository project aims at becoming a central searching point for pattern related information. By gathering feedback from the pattern community, the following key concerns were identified.

1. **Community support**: The repository should be open for contribution and should motivate users to participate, in order to create a vivid community.
2. **Functionality**: It is important that the repository offers a good way to explore patterns. It has to provide search functionality for patterns which incorporates all aspects of patterns (forces, context, relations etc.). The repository should be context-aware that means that it should display related patterns or categories. All in all it should support the process of making design decisions by finding appropriate patterns.
3. **Usability**: The repository must provide an intuitive interface, with main focus on the process of adding, editing and finding patterns.
4. **Moderation**: The repository should offer a possibility for quality rating. Although everybody can contribute pattern to the repository, there will different means to rate and improve the quality of the stored patterns.
5. **Copyright:** The repository focusses on content that is actively shared with the community, but it will also offer a way to deal with proprietary content and different licensing models.

A unique feature that the OPR provides, is a new approach to capture and archive design decisions. Furthermore it provides a possibility to be integrated with CASE-tool. Respectively, it could be integrated in any service or program.

2.3 User characteristics

User The common user is registered with the system. He uses the system to manage patterns and to document his design decisions. The authentication is needed to associate patterns with an author and to select a license model. Furthermore, it enables the user to make use of the Design Decision Management.

Guest Basically, the guest uses the system to retrieve information about patterns. He will use the search and browsing functionality to find the information he is looking for. However, some of the guests will edit and add pattern descriptions. His motivation not to register is anonymity.

Moderator The moderator is a user with extended rights. He is familiar with pattern descriptions and uses the system to ensure quality within the repository. He is concerned with reviewing pattern descriptions and rating their quality.
In addition he is able to manage patterns on a higher level, like deleting a pattern or category.

Administrator The administrator manages the system, which includes user management and data management.

The user groups of the system are hierarchical structured that means an administrator is at the same time a guest, depending on the task he is doing (s. Figure 1).

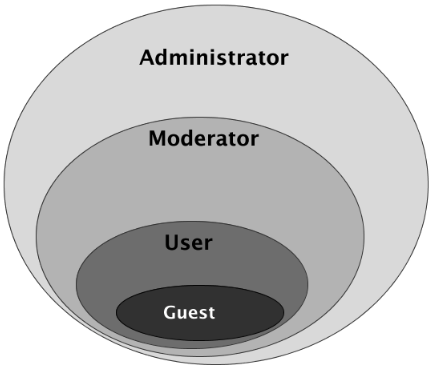


Figure 1: Inheritance of user rights

2.4 Product functions

According to the three parts identified in section 1.2, the use case model is broken into packages. Additionally the configuration package contains overall configuration use cases. The use cases presented in this survey are high level use cases that are refined in the Specific Requirements section. Please refer to Appendix A2 for a Traceability Matrix, that clarifies the relationships between high level use cases and specific requirement use cases.

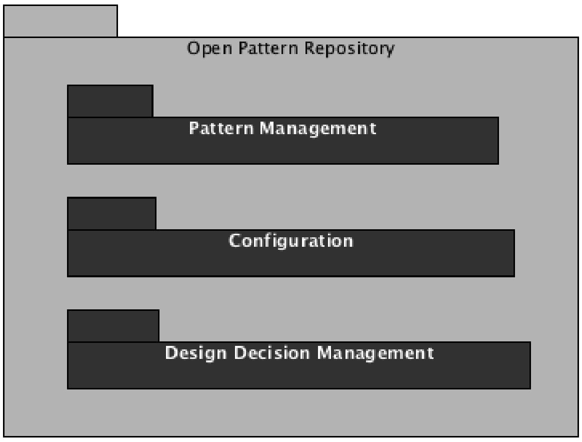


Figure 2: Use case packages

2.4.1 Package - Pattern Management

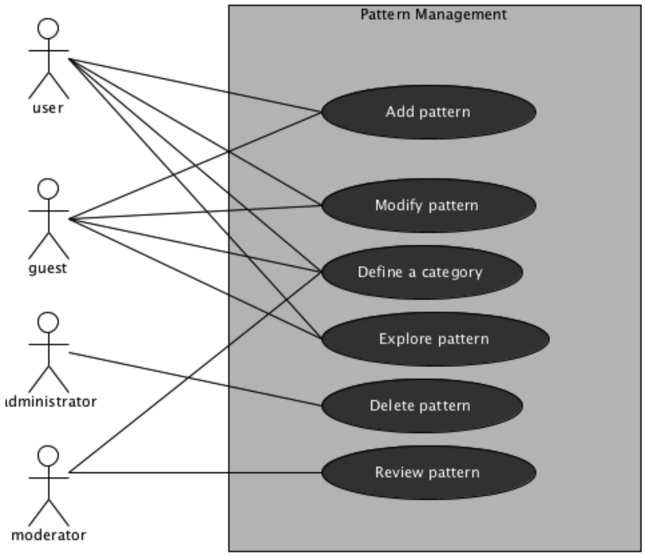


Figure 3: Relation of actors and use cases in package Pattern Management

Code:	HLUC-1	Name:	Add Pattern	Package:	Pattern Management
Description:	In this use case the user adds a pattern. It is intended for guests and users. The pattern descriptions are the basis of the OPR and represent the content. Every pattern shall be represented by a name (e.g. Publish-and-Subscribe) and a description. It shall be possible to declare at least the context, problem and solution. Furthermore, it shall be possible to declare source, categories and tags but *none of these values is mandatory*. Furthermore it shall be possible to *manage relationships* with other patterns and to associate artifacts with the pattern. Due to copyright issues it shall be possible to *add licensing and copyright information* for free text (description, solution etc.) and each artifact. It shall be possible to *add a pattern wizard-based or with a quick-add function*. In both cases the process should be *simple and intuitive*. The wizard shall provide different templates for different types of patterns and technologies.				
Refined by:	SPUC-1.1 - Add Pattern wizard-based, SPUC-1.2 - Quick add Pattern				
Version:	1	Author:	cm		

Table 1: HLUC-1 - Add Pattern

Code:	HLUC-2	Name:	Modify Pattern	Package:	Pattern Management
Description:	In this use case the user modifies a description. It is intended for guests and users. Each modification shall be *stored in a new version* and all previous versions shall be available for comparison and backtracking. The system should display a preview version at each time of modification. According to the licensing information the possibility for *modification of original descriptions differ*. For instance if a restrictive license has been chosen, only the author can modify the description. All other user have to add a new version based on their own text. Please refer to Appendix A1.2 for further information. Only authenticated users can select a restrictive license which prohibits the modification. On the other hand tagging of pattern is allowed by anyone. New tags for patterns are not stored in a new version. This use case also includes removal of a pattern.				
Refined by:	SPUC-2.1 - Modify Pattern, SPUC-2.2 - Propose to delete a Pattern, SPUC-2.3 - Delete a Pattern				
Version:	3	Author:	cm, mv		

Table 2: HLUC-2 - Modify Pattern

Code:	HLUC-3	Name:	Delete Pattern	Package:	Pattern Management
Description:	In this use case the administrator removes a description. Due to copyright issues it shall be possible to delete a pattern and all associated data, as well as a specific version of a pattern or a specific artifact, irreversible.				
Refined by:	SPUC-3.1 - Remove Pattern irrevocable				
Version:	1	Author:	cm		

Table 3: HLUC-3 - Delete Pattern

Code:	HLUC-4	Name:	Explore Patterns	Package:	Pattern Management
Description:	In this use case the user can explore the patterns in the repository by browsing, searching or viewing a pattern. It is intended for guests and users. It shall be possible to browse the repository *by category* and to *follow the relationships* between patterns. It should also be possible to browse not only by category but by associated tags. The system shall provide advanced search which scans for the *name, description, solution, context, problem and relationships*. The average and maximum time to search shall meet a specified value. It shall also be possible to enter a *unique pattern identifier* to reach the pattern directly. This is a convenient way for reaching pattern quickly.				
Refined by:	SPUC-4.1 - Browse Pattern, SPUC-4.2 - Search for Pattern, SPUC-4.3 - View Pattern, SUPR-Q4, SUPR-Q5				
Version:	1	Author:	cm		

Table 4: HLUC-4 - Explore Patterns

Code:	HLUC-5	Name:	Review a Pattern	Package:	Pattern Management
Description:	In this use case the moderator reviews a description. It shall be possible to review and rate a description for reasons of quality assurance. Possible values are *unreviewed, pending or reviewed.* The ratings are used as an indicator for the quality of the pattern description. All patterns that are initially added have the status “unreviewed”.				
Refined by:	SPUC-5.1 - Review Pattern				
Version:	1	Author:	cm		

Table 5: HLUC-5 - Review a Pattern

Code:	HLUC-6	Name:	Define a category	Package:	Pattern Management
Description:	In this use case a user creates, deletes and edits categories. Users and guests are able to create new categories but editing and deleting is restricted to moderators. Categories are organized in a tree-like structure with the category “all” as the root. A pattern could be a member of several categories. While entering a new pattern, it is important that the system makes suggestions about existing categories and motivates the user to use an existing category instead creating a new one. It should be possible to transfer a complete category or its contents to another category. It shall be possible to delete a category. If a category is deleted its members will automatically become a direct member of the parent category, only if it is not an indirect member of the parent category at the same time (s. Appendix A3).				
Refined by:	SPUC-6.1 - Add Category, SPUC-6.2 - Edit Category, SPUC-6.3 - Remove Category, SPUC-6.4 - Transfer Category				
Version:	2	Author:	cm, mv		

Table 6: HLUC-6 - Define a category

2.4.2 Package - Configuration

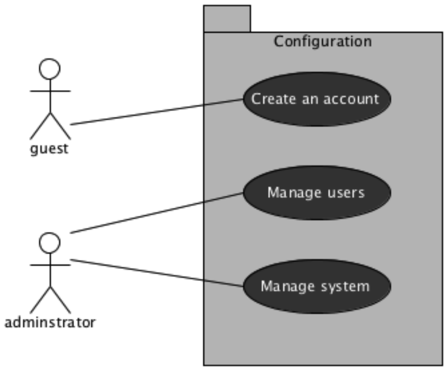


Figure 4: Relation of actors and use cases in package Configuration

Code:	HLUC-7	Name:	Create an account	Package:	Pattern Management
Description:	In this use case a guest creates a user account. The user has to fill in mandatory information like *screen name, email address and password*, but should have the *possibility to store further information* like real name and contact information. After entering at least the mandatory information the system *validates the users input* to avoid misuse. This should be done by e-mail validation. As soon as the validation is successful the user can log into the system as user. It may be possible to log in with an OpenID.				
Refined by:	SPUC-7.1 - Create an account				
Version:	2	Author:	mv		

Table 7: HLUC-7 - Create an account

Code:	HLUC-8	Name:	Manage users	Package:	Configuration
Description:	In this use case an OPR administrator *configures user accounts and user rights.* User accounts can be accounts from OPR users, moderators or administrators. The account can be moved into another user group like moderator or administrator. This use case includes *adding, browsing, changing and removing user accounts.*				
Refined by:	SPUC-8.1 - Manage user rights				
Version:	2	Author:	mv		

Table 8: HLUC-8 - Manage users

2.4.3 Package - Design Decision Management

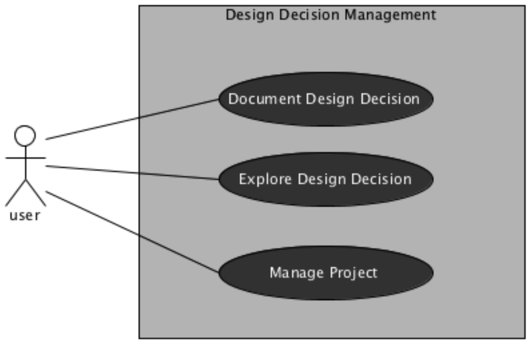


Figure 5: Relation of actors and use cases in package Design Decision Management

Code:	HLUC-9	Name:	Document Design Decision	Package:	Design Decision Management
Description:	In this use case an OPR user documents his design decision. The user has to authenticate to the system in order to document. After starting the documentation process the user is *guided through a wizard based questionary to add his design decision*. The user has to choose a project for storing the design decision. Design decisions are stored by referencing generic data from the pattern repository. This makes the documentation of decisions easier and relieves the documenters work. While processing, the system shall provide a possibility of storing links to external files, which help understanding the design decision and the rationale behind it. These can be UML diagrams for example. Furthermore the system shall provide a functionality to document the forces which led the user to his decision. Architects shall be able to revoke design decisions. These could be done, when a better solution to the problem has been discovered. Design decisions can not be removed from the system in order to preserve architectural knowledge and decision rationale.				
Refined by:	SPUC-9.1 - Add design decision, SPUC-9.2 - Edit design decison, SPUC-9.3 - Revoke design decision				
Version:	3	Author:	mv, cm		

Table 9: HLUC-9 - Document Design Decision

Code:	HLUC-10	Name:	Explore Design Decisions	Package:	Design Decision Management
Description:	In this use case an OPR user explores a design decision. The user has to authenticate to the system and selects a project to see a list of his design decisions. After choosing a design decision a *detailed page shows up generated by information from the online repository and links to stored files*. This page shows the stored drivers, as well as a list of pattern which have a relationship. If desired the user may be able to choose alternatives to his design decision. These are generated by the online repository. The system should provide functionality to generate a printable document of design decisions. As file type a human readable and editable format like rich text format shall be used . It should support PDF and may export the document as XML.				
Refined by:	SPUC-10.1 - Browse design decisions, SPUC-10.2 - View design decision, SPUC-10.3 - Generate Project Documentation				
Version:	1	Author:	mv		

Table 10: HLUC-10 - Explore Design Decisions

Code:	HLUC-11	Name:	Manage Project	Package:	Design Decision Management
Description:	In this use case an OPR user manages a project for organizing design decisions. Defining includes *Adding, Editing* and *Removing* a project. Design decisions are made during software projects, hence they have a relation to a specific project. Each project has a name and a description for unique identification. Projects can be browsed and list all containing design decisions. Additionally, the user shall be able to grant OPR users access to his project.				
Refined by:	SPUC-11.1 - Add Project, SPUC-11.2 - Edit Project, SPUC-11.3 - Remove Project				
Version:	2	Author:	mv		

Table 11: HLUC-11 - Manage Project

2.5 General contraints

- The complete software and documentation shall be published under GPL v3 license.

- The application shall be open for everyone. That means:
  - Not restricted to any countries.
  - Editable and Viewable by everyone.
  - Does not require any special programs to use the OPR.
  - Does not require registration costs.
  - Is based on accepted web-standard.
  - It is open source.

2.6 Assumptions and dependencies

none

3 Specific requirements

3.1 Use Cases

3.1.1 Package - Pattern Management

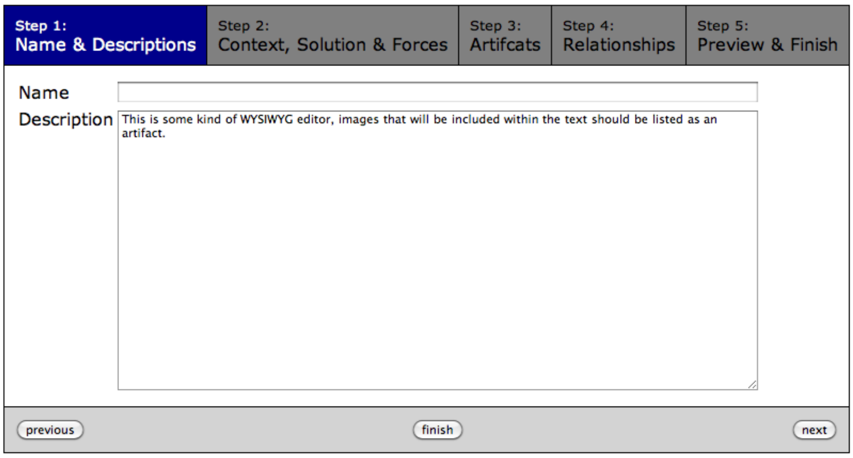
Code:	SPUC-1.1	Name:	Add Pattern wizard-based	Package:	Pattern Management
Actors:	Guest				
Goal:	User wants to add a pattern to repository.				
Preconditions:	-				
Basic flow:	1. This use case starts when the user selects *Add Pattern* 2. The user selects the template which he wants to use. The template defines the available fields, the accepted values and the flow of the wizard. 3. The user will be led by the wizard through the steps. The *name, context, problem, solution and licensing information* shall be filled in first, so that it is possible to *finish the wizard at any time*. Furthermore it shall be possible to *navigate forth and back* through the steps. It is mandatory that the system *checks the validity of all data* before the user moves on to the next step and informs him about the conflict. However, the wizard *should not preclude the user from navigating* but instead it should display a summary of conflicts. The wizard will ask amongst others for source, categories, tags, artifacts and relationships to other patterns. A relationship could be a variant, alternative or combination. 4. The user select finish. The system displays a summary of the pattern. 5. The user confirms the pattern. 6. The system stores the pattern as the first version and establishes all links to other patterns.  Step 1: Name & Descriptions Step 2: Context, Solution & Forces Step 3: Artifcats Step 4: Relationships Step 5: Preview & Finish previous finish next Name Description: This is some kind of WYSIWYG editor, images that will be included within the text should be listed as an artifact. A larger version of the illustration can be found in Figure 6				
Variations:	-				
Postconditions:	-				
Version:	3	Author:	cm, mv		

Table 12: SPUC-1.1 - Add Pattern wizard-based

Code:	SPUC-1.2	Name:	Quick add Pattern	Package:	Pattern Management
Actors:	Guest				
Goal:	User wants to add patterns to the repository				
Preconditions:	-				
Basic flow:	1. This use case starts when the user selects *Quick add Pattern* 2. The user selects the template which he wants to use. The template defines the available fields. 3. The system displays a single form with all fields. The form offers support for copy&paste. 4. The user fills the form. 5. The user selects finish. The system displays a summary of the pattern. 6. The user confirms the pattern. 7. The system stores the pattern as the first version.				
Variations:	A registered user should have the possibility of storing an unfinished pattern at any time.				
Postconditions:	-				
Version:	3	Author:	mv, cm		

Table 13: SPUC-1.2 - Quick add Pattern

Code:	SPUC-2.1	Name:	Modify Pattern	Package:	Pattern Management
Actors:	Guest				
Goal:	User wants to modify a pattern.				
Preconditions:	-				
Basic flow:	1. This use case starts when the user selects *Edit* pattern. 2. The system displays the pattern in an editable form. 3. The user changes the data and select finish. During his changes the user should have the possibility to preview his changes in a detailed view as mentioned in SPUC-4.3 - View Pattern. 4. The system validates the user input and creates a new version. The previous version will be archived.				
Variations:	2.1 If the pattern has been published with a restrictive license and the editor is not the author of the pattern. The system displays a blank form and informs the user that he's not allowed to use the previous version as a basis due to copyright restrictions. The flow continues with step 3.				
Postconditions:	The pattern description changed.				
Version:	3	Author:	cm, mv		

Table 14: SPUC-2.1 - Modify Pattern

Code:	SPUC-2.2	Name:	Propose to delete a Pattern	Package:	Pattern Management
Actors:	Guest				
Goal:	User wants to make a proposal to delete a pattern because it is obsolete or duplicated				
Preconditions:	-				
Basic flow:	• This use case starts when the selects *Propose for delete* at the pattern he wants to delete. • The system prompts the user to enter a reason. It also offers a list of predefined reason such as duplicate, obsolete. • The user confirms to propose the pattern for deletion. • The system stores the proposal and displays a success message.				
Variations:	-				
Postconditions:	The Pattern is proposed for deletion.				
Version:	1	Author:	cm		

Table 15: SPUC-2.2 - Propose to delete a Pattern

Code:	SPUC-2.3	Name:	Delete a Pattern	Package:	Pattern Management
Actors:	Moderator				
Goal:	Moderator deletes a proposed pattern.				
Preconditions:	-				
Basic flow:	• This use case starts when the user selects *Proposed patterns*. • The systems displays a list of pattern that were proposed for deletion. A summary of each pattern along with the reason is displayed. • The user selects *acceppt* or *refuse* for each pattern and select *process proposals*. • The system displays a list of patterns that will be deleted and prompts the user to confirm. The patterns are only marked as deleted and will no longer show up in the repository. However, the administrator can reactivate them at any time. • The user confirms and the systems marks the pattern as deleted.				
Variations:	-				
Postconditions:	The Pattern is deleted.				
Version:	1	Author:	cm		

Table 16: SPUC-2.3 - Delete a Pattern

Code:	SPUC-3.1	Name:	Remove Pattern irrevocable	Package:	Pattern Management
Actors:	Administrator				
Goal:	Administrator wants to remove a pattern irreversible from the repository. For example because of copyright violations.				
Preconditions:	Authenticated user is administrator				
Basic flow:	1. This use case starts when the administrator selects *Remove Pattern* on the pattern he wants to be removed. 2. The system asks, if the whole content should be removed or only parts of it like the description, forces, context, solution, artifacts etc.. 3. The administrator makes his selection and selects *Remove*. 4. The system *prompts the administrator to confirm his choice*. 5. The administrator confirms his choice. 6. The system removes the selected parts of the stored pattern.				
Variations:	-				
Postconditions:	A pattern or parts of it are removed.				
Version:	1	Author:	mv		

Table 17: SPUC-3.1 - Remove Pattern irrevocable

Code:	SPUC-3.2	Name:	Remove a version of a Pattern	Package:	Pattern Management
Actors:	Administrator				
Goal:	The administrator wants to remove a specific version of a pattern from the repository.				
Preconditions:	Authenticated user is administrator				
Basic flow:	1. This use case starts when the administrator selects *View Pattern Versions* on a pattern he wants to be removed 2. The system provides an overview of the pattern versions. 3. The administrator selects one or more pattern versions for deletion and confirms with *Remove Versions*. 4. The system asks for each version, if the whole content should be removed or only parts of it like the description, forces, context, solution, artifacts etc.. 5. The administrator makes his selection and selects *Remove*. 6. The system *prompts the administrator to confirm his choice*. 7. The administrator confirms his choice. 8. The system removes the selected parts of the stored pattern versions or selected artifacts.				
Variations:	-				
Postconditions:	One or several patterns or parts of them are removed.				
Version:	1	Author:	mv		

Table 18: SPUC-3.2 - Remove a version of a Pattern

Code:	SPUC-4.1	Name:	Browse Pattern	Package:	Pattern Management
Actors:	Guest				
Goal:	User wants to browse the pattern repository				
Preconditions:	-				
Basic flow:	1. This use case starts when the user selects browse. 2. The system provides an overview of pattern categories and tags. The system may provide a tag-cloud for simplified navigation. 3. The user selects one ore more of these subdivisions. The system should provide a list view of these subdivisions. The system may also provide a tree-like structure for browsing the subdivisions. 4. The system provides the results which match the parameters.				
Variations:	-				
Postconditions:	The user found the pattern he wants to take a look at.				
Version:	4	Author:	mv, cm		

Table 19: SPUC-4.1 - Browse Pattern

Code:	SPUC-4.2	Name:	Search for Pattern	Package:	Pattern Management
Actors:	Guest				
Goal:	The user wants to search for a pattern.				
Preconditions:	-				
Basic flow:	1. This use case starts when the user selects *Search Pattern*. 2. The system displays a search form and prompts the user to enter his search data. 3. The user adds the search values he is looking for. It shall be possible to search all properties of a pattern. 4. The user selects *Search*. 5. The system displays matching pattern.				
Variations:	5.1 If no matching patterns were found the system informs the user and prompts to change the search conditions. The system may provide a collection of patterns with the closest match of search parameters.				
Postconditions:	-				
Version:	2	Author:	mv, cm		

Table 20: SPUC-4.2 - Search for Pattern

Code:	SPUC-4.3	Name:	View Pattern	Package:	Pattern Management
Actors:	Guest				
Goal:	The user wants to view a specific pattern.				
Preconditions:	-				
Basic flow:	1. This use case starts when the user has searched for a pattern or browsed a pattern. 2. The user selects *View Pattern*. 3. The system provides an overview of the description and artifacts. It shall be possible to see patterns which have relationships to the current pattern and switch to them easily. The system displays the newest version of the pattern but in the case that an older version has been reviewed it will inform the user that a *previous version has been reviewed along with the rating*.				
Variations:	1.1 This use case starts when the user enter a unique pattern ID. 1.2 The system looks up the pattern and displays the pattern. The flow continues with step 3. 3.1 The user wants to see another version of the pattern and selects *Show Pattern History*. 3.2 The system provides a versioning overview of the current pattern. It may also provide authors comments on the different versions, since this facilitates the finding of a specific pattern version.				
Postconditions:	The user sees a pattern with its related description and artifacts.				
Version:	3	Author:	mv, cm		

Table 21: SPUC-4.3 - View Pattern

Code:	SPUC-5.1	Name:	Review Pattern	Package:	Pattern Management
Actors:	Moderator				
Goal:	The moderator wants to review and rate a pattern description.				
Preconditions:	Authenticated user is moderator				
Basic flow:	1. This use case starts when the moderator views a pattern. See use case SPUC-4.3 - View Pattern. 2. The moderator reads the content of the description and figures out its quality. All new added patterns have the rating status *unreviewed.* 3. The moderator selects one of the reviewing status *pending or reviewed.* Pending means that the description or artifacts do not match the quality standards. In this case the moderator could edit the pattern on his own (if this is possible). Communication between the reviewer and the author of the pattern is not within the scope of the system. 4. The system saves the changed rating status and informs the moderator.				
Variations:	-				
Postconditions:	A pattern is rated.				
Version:	3	Author:	mv, cm		

Table 22: SPUC-5.1 - Review Pattern

Code:	SPUC-6.1	Name:	Add Category	Package:	Pattern Management
Actors:	User				
Goal:	The user wants to add a category				
Preconditions:	-				
Basic flow:	1. This use case starts when the user selects *Add Category.* 2. The user enters the name and chooses a parent category. 3. The user selects *Add.* 4. The system searches for categories with the same name. 5. The system stores the category.				
Variations:	4.1 If the system finds a category with a similar name it displays a list of the found patterns. 4.2 The system shows a dialog that a similar pattern already exists and asks the user if he really wants to add the category.				
Postconditions:	A category has been added.				
Version:	1	Author:	mv, cm		

Table 23: SPUC-6.1 - Add Category

Code:	SPUC-6.2	Name:	Edit Category	Package:	Pattern Management
Actors:	Moderator				
Goal:	The moderator wants to edit a category				
Preconditions:	Authenticated user is moderator				
Basic flow:	1. This use case starts when the user selects *Manage Categories*. 2. The system displays a list of categories in a tree-like order. 3. The user selects the category he wants to change. 4. The system displays the category name in an editable form. 5. The user changes the name and selects *Change*. 6. The system stores the change.				
Variations:	-				
Postconditions:	A category has been modified.				
Version:	1	Author:	mv, cm		

Table 24: SPUC-6.2 - Edit Category

Code:	SPUC-6.3	Name:	Remove Category	Package:	Pattern Management
Actors:	Moderator				
Goal:	The moderator wants to remove a category				
Preconditions:	Authenticated user is moderator				
Basic flow:	1. This use case starts when the user selects *Manage Categories*. 2. The system displays a list of categories in a tree-like order. 3. The user selects the category he wants to remove. 4. The system asks for confirmation. 5. The user confirms the deletion. 6. The system removes the category. If the category contains patterns these get automatically assigned to the parent category.				
Variations:	-				
Postconditions:	A category has been removed.				
Version:	1	Author:	mv, cm		

Table 25: SPUC-6.3 - Remove Category

Code:	SPUC-6.4	Name:	Transfer Category	Package:	Pattern Management
Actors:	Moderator				
Goal:	The moderator wants to transfer a category				
Preconditions:	Authenticated user is moderator				
Basic flow:	1. This use case starts when the user selects *Manage Categories*. 2. The system displays a list of categories in a tree-like order. 3. The user selects the category he wants to transfer. 4. The system asks if the pattern within the category should be transferred or the category itself. 5. The user selects the new parent category. 6. The system transfers the category.				
Variations:	-				
Postconditions:	A category has been transferred.				
Version:	3	Author:	mv, cm		

Table 26: SPUC-6.4 - Transfer Category

3.1.2 Package - Configuration

Code:	SPUC-7.1	Name:	Create an account	Package:	Pattern Management
Actors:	Guest				
Goal:	A guest user wants to create an OPR account.				
Preconditions:	-				
Basic flow:	1. This use case starts when the user selects *Create User Account*. 2. The system displays a form which has to be filled with *at least screen name, email address and password*. Other possible information could be *real name, postal address, telephone number etc.*. 3. The user enters his data and selects *Create*. 4. The system validates the email address for correctness and stores the user information. The email validation should be safer, for example by sending an confirmation email to the entered address to validate the user. 5. The system should send an email with an activation key. The user has to enter this key in order to activate his account and verify the correctness of his email address. 6. The system informs the user of the account creation. 7. The user can log into the system using his credentials.				
Variations:	0.1 It may also be possible to log into the system using the OpenID system. This would simplify the legitimation process and make registering more convenient for the user.				
Postconditions:	The user has an OPR account and is able to login into the system.				
Version:	1	Author:	mv		

Table 27: SPUC-7.1 - Create an account

Code:	SPUC-8.1	Name:	Manage user rights	Package:	Configuration
Actors:	Administrator				
Goal:	The administrator wants to change users permissions.				
Preconditions:	Authenticated user is administrator, User has no special permissions within the OPR				
Basic flow:	1. This use case starts when the administrator selects *manage user right* for a specific user. 2. The system displays an overview of the user account. Possible user rights to be set are *Moderator* and *Administrator*. 3. The administrator sets or changed the desired user rights and selects *Save*. 4. The system saves the changes and displays, that the changes were successful.				
Variations:	-				
Postconditions:	An user has new permissions.				
Version:	1	Author:	mv		

Table 28: SPUC-8.1 - Manage user rights

3.1.3 Package - Design Decision Management

Code:	SPUC-9.1	Name:	Add design decision	Package:	Design Decision Management
Actors:	Authenticated user				
Goal:	A user wants to document a design decision				
Preconditions:	User is authenticated				
Basic flow:	1. This use case starts when the user selects *Add Design Decision*. 2. The user selects the project. 3. The user will be led by a wizard through the steps. He has to select which pattern he used within the design decision and which forces lead him to his decision. The user shall be able to link to specific artifacts, which help to understand the decision of the architect. These links are textfields that does not limit the type of the linkage. For example it can refer to a file of the users intranet or an internet url. It should also be possible to store revisions of files or files, that were affected by the design decision. Furthermore, the wizard will ask for context, assumptions, related requirements and time time when decision has been made. It shall be possible to *abort the wizard at any time* during the documentation phase to improve the users convenience. 4. After the design decision has been documented successfully the user sees an overview of his stored design decisions.				
Variations:	2.1 If the user decides to abort the wizard, he should have the possibility to continue at a later point.				
Postconditions:	A design decision is documented.				
Version:	2	Author:	mv,cm		

Table 29: SPUC-9.1 - Add design decision

Code:	SPUC-9.2	Name:	Edit design decison	Package:	Design Decision Management
Actors:	Authenticated user				
Goal:	A user wants to edit a stored design decision				
Preconditions:	User is authenticated, User has stored design decisions				
Basic flow:	1. This use case starts when the user selects *Design Decision Management.* 2. The system presents the user's stored design decisions. 3. The user chooses a design decision and selects *Edit.* 4. The system shows a detailed *editable view* of the design decision. 5. The user changes the design decision to match his needs. 6. The user selects *Save.* 7. The system saves the changes to the design decision and informs the user about the successful change.				
Variations:	-				
Postconditions:	A design decision is edited.				
Version:	1	Author:	mv		

Table 30: SPUC-9.2 - Edit design decison

Code:	SPUC-9.3	Name:	Revoke design decision	Package:	Design Decision Management
Actors:	Authenticated user				
Goal:	A user wants to revoke a design decision				
Preconditions:	User is authenticated, User has stored design decisions				
Basic flow:	1. This use case starts when the user selects *Design Decision Management.* 2. The system presents the user's stored design decisions. 3. The user chooses a design decision and selects *Revoke.* 4. The user has to enter a reason for the revocation. 5. The system displays a warning and *claims for an acknowledgement.* 6. The user confirms the message. 7. The system marks the design decision as revoked.				
Variations:	-				
Postconditions:	A design decision is revoked				
Version:	2	Author:	mv,cm		

Table 31: SPUC-9.3 - Revoke design decision

Code:	SPUC-10.1	Name:	Browse design decisions	Package:	Design Decision Management
Actors:	Authenticated user				
Goal:	A user wants to browse his design decisions				
Preconditions:	User is authenticated, User has stored design decisions				
Basic flow:	1. This use case starts when the user is logged into the system. 2. The user selects a project 3. The system presents the design decisions that are associated with the project.				
Variations:	-				
Postconditions:	-				
Version:	2	Author:	mv,cm		

Table 32: SPUC-10.1 - Browse design decisions

Code:	SPUC-10.2	Name:	View design decision	Package:	Design Decision Management
Actors:	Authenticated user				
Goal:	A user wants to take a look at a design decision				
Preconditions:	User is authenticated, User has stored design decisions				
Basic flow:	1. This use case starts when the user selects *Show Design Decision* on one of the stored design decisions. 2. The system provides an *overview of the design decision*. This overview shall provide a clean and good view of the design decisions. The system shall also provide alternatives to the chosen design decision as well as outline the quality attributes and consequences of the current design decision. Furthermore it shall be possible to see patterns which provide a good combination with the design decision.				
Variations:	-				
Postconditions:	-				
Version:	1	Author:	mv		

Table 33: SPUC-10.2 - View design decision

Code:	SPUC-10.3	Name:	Generate Project Documentation	Package:	Design Decision Management
Actors:	Authenticated user				
Goal:	The user wants to generate a printable documentation				
Preconditions:	User is authenticated				
Basic flow:	1. This use case starts when the user selects a *project*. 2. The user select *generate documentation* 3. The system displays a form with exporting options like filetype and filename. 4. The user select *generate*. 5. The system generates the documentation. The document will start with the project name as a header and then lists the design decision in chronologic order. 6. The system offers the document as a download.				
Variations:	-				
Postconditions:	A project specific documentation has been generated.				
Version:	1	Author:	cm		

Table 34: SPUC-10.3 - Generate Project Documentation

Code:	SPUC-11.1	Name:	Add Project	Package:	Design Decision Management
Actors:	Authenticated user				
Goal:	The user wants to add a project to store design decisions.				
Preconditions:	User is authenticated				
Basic flow:	1. This use case starts when the user selects *Add Project* 2. The system displays a form where the user can add a name and a description for the project. 3. The user enters his information on project name and optional description. The user shall also have the possibility to give other authenticated users of the OPR access to a project. Thus other users, for example architects, who have to maintain a system can take a look at the previous design decisions. 4. The user selects *Save*. 5. The system stores the project information.				
Variations:	-				
Postconditions:	A project for storing design decisions has been created.				
Version:	2	Author:	mv		

Table 35: SPUC-11.1 - Add Project

Code:	SPUC-11.2	Name:	Edit Project	Package:	Design Decision Management
Actors:	Authenticated user				
Goal:	The user wants to change a projects properties.				
Preconditions:	User is authenticated				
Basic flow:	1. This use case starts when the user selects *Edit Project*. 2. The system displays the stored project information (name and description). 3. The user edits the name, description and allowed users. 4. The user selects *Save*. 5. The system stores the changes data.				
Variations:	-				
Postconditions:	Name, description and allowed users for a project have been changed.				
Version:	1	Author:	mv		

Table 36: SPUC-11.2 - Edit Project

Code:	SPUC-11.3	Name:	Remove Project	Package:	Design Decision Management
Actors:	Authenticated user				
Goal:	The user wants to remove a project.				
Preconditions:	User is authenticated				
Basic flow:	1. This use case starts, when the user selects *Remove Project*. 2. The system prompts the user if he wants to delete the whole project with its stored design decisions or only the project. 3. The user selects *Remove design decisions too*. 4. The system removes the project and all containing design decisions. 5. The system notifies, that the project has been deleted.				
Variations:	3.1 The user selects *Only remove project*. 3.2 The system removes the project. 3.3 The flow continues at step 5.				
Postconditions:	A whole project with its containing design decisions has been deleted.				
Version:	1	Author:	mv		

Table 37: SPUC-11.3 - Remove Project

3.2 Supplementary Requirements

3.2.1 Interface Requirements

Code:	SUPR-I1	Description:	The application shall provide a web interface which can be accessed by any web browser without the need for plugins.
Version:	3	Author:	cm, mv

Table 38: SUPR-I1

Code:	SUPR-I2	Description:	The application shall provide webservices. The webservices do not provide all functionality. It supports pattern management, but does not support design decisions management and configuration. In detail it shall support SPUC-1.2 - Quick add Pattern, SPUC-2.1 - Modify Pattern, SPUC-2.2 - Propose to delete a Pattern, SPUC-4.1 - Browse Pattern, SPUC-4.3 - View Pattern
Version:	2	Author:	cm

Table 39: SUPR-I2

3.2.2 Software System Attributes

3.2.3 Usability

These are the usability requirements identified for the OPR. For an detailed description about verifiability, goals and measurement please refer to Appendix A5

Understandibility

Code:	SUPR-Q1	Description:	The Interface elements should be easy to understand
Version:	1	Author:	cm

Table 40: SUPR-Q1

Learnability

Code:	SUPR-Q2	Description:	The help should be context sensitive and explain how to achieve common tasks.
Version:	1	Author:	cm

Table 41: SUPR-Q2

Code:	SUPR-Q3	Description:	The system should be easy to learn.
Version:	1	Author:	cm

Table 42: SUPR-Q3

Operability

Code:	SUPR-Q4	Description:	The interface actions and elements should be consistent.
Version:	1	Author:	cm

Table 43: SUPR-Q4

Code:	SUPR-Q5	Description:	Error messages should explain how to recover from the error.
Version:	1	Author:	cm

Table 44: SUPR-Q5

Code:	SUPR-Q6	Description:	Actions which cannot be undone should ask for confirmation.
Version:	1	Author:	cm

Table 45: SUPR-Q6

Attractiveness

Code:	SUPR-Q7	Description:	The screen layout and color should be appealing
Version:	1	Author:	cm

Table 46: SUPR-Q7

3.2.4 Reliability

Code:	SUPR-Q8	Description:	The system shall guarantee a data integrity of 100%
Version:	1	Author:	cm

Table 47: SUPR-Q8

Code:	SUPR-Q9	Description:	The system shall guarantee a recoverability of 100%
Version:	1	Author:	cm

Table 48: SUPR-Q9

3.2.5 Performance

Code:	SUPR-Q10	Description:	The average time to receive search results shall be 2,5 seconds.
Version:	2	Author:	cm, mv

Table 49: SUPR-Q10

Code:	SUPR-Q11	Description:	The maximum time to receive search results shall be 5,0 seconds.
Version:	2	Author:	cm, mv

Table 50: SUPR-Q11

3.2.6 Extensibility

Code:	SUPR-Q12	Description:	The system shall provide mechanisms for expanding/enhancing the system with new capabilities without having to make major changes to the system infrastructure.
Version:	1	Author:	cm

Table 51: SUPR-Q12

3.2.7 Availability

Code:	SUPR-Q13	Description:	The system shall be designed in such a way that the intended period of operation is above 99.9%. This corresponds roughly to 9 hours downtime within a year.
Version:	3	Author:	cm, mv

Table 52: SUPR-Q13

3.2.8 Scalability

Code:	SUPR-Q14	Description:	The system shall be scalable in the amount of patterns that can be stored in the repository.
Version:	1	Author:	cm

Table 53: SUPR-Q14

Code:	SUPR-Q15	Description:	The system shall be scalable in the amount of concurrent users that are retrieving and updating data from the repository.
Version:	1	Author:	cm

Table 54: SUPR-Q15

3.2.9 Portability

Code:	SUPR-Q16	Description:	The system shall be portable to any major operating system and server operating system without recompiling the sources.
Version:	1	Author:	cm

Table 55: SUPR-Q16

3.2.10 Security

Code:	SUPR-Q17	Description:	Password shall not be displayed in plain text.
Version:	1	Author:	cm

Table 56: SUPR-Q17

Code:	SUPR-Q18	Description:	Administrative actions like account managing shall be logged.
Version:	1	Author:	cm

Table 57: SUPR-Q18

Code:	SUPR-Q19	Description:	All pattern modifications shall log author, time, date and IP
Version:	1	Author:	cm

Table 58: SUPR-Q19

Code:	SUPR-Q20	Description:	Login information shall be transfered encrypted.
Version:	1	Author:	mv

Table 59: SUPR-Q20

References

[1] F. Buschman, R. Meunier, H. Rohnert, P. Sommerland, and M. Stal. *Pattern-Oriented Software Architecture - A system of patterns*. Number 0-471-95869-8. Wiley, 1996.

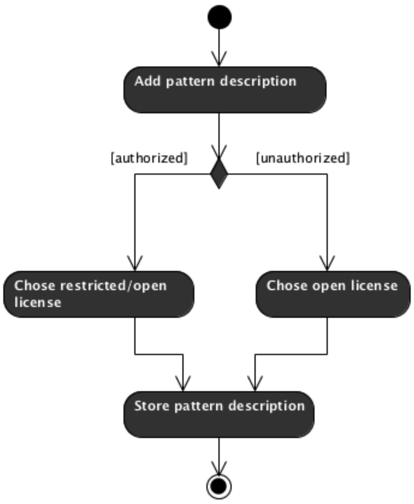
[2] M. de Jong. Identify and preserve reusable architectural knowledge from technologies. Technical report, University of Groningen.

[3] IEEE. Recommended practice for software requirements specifications. Technical report, IEEE, 1998.

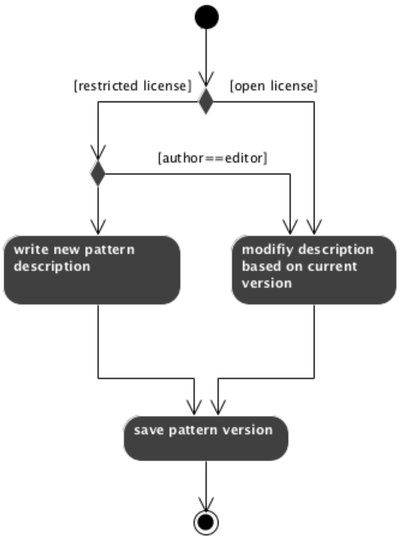
Appendix

A1 Copyright handling

A1.1 Add pattern description



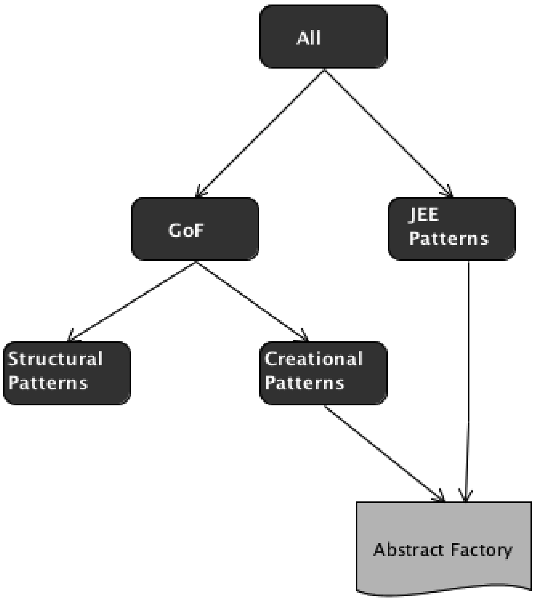
A1.2 Edit pattern description


[restricted license]
[open license]
[author==editor]
write new pattern description
modifiy description based on current version
save pattern version

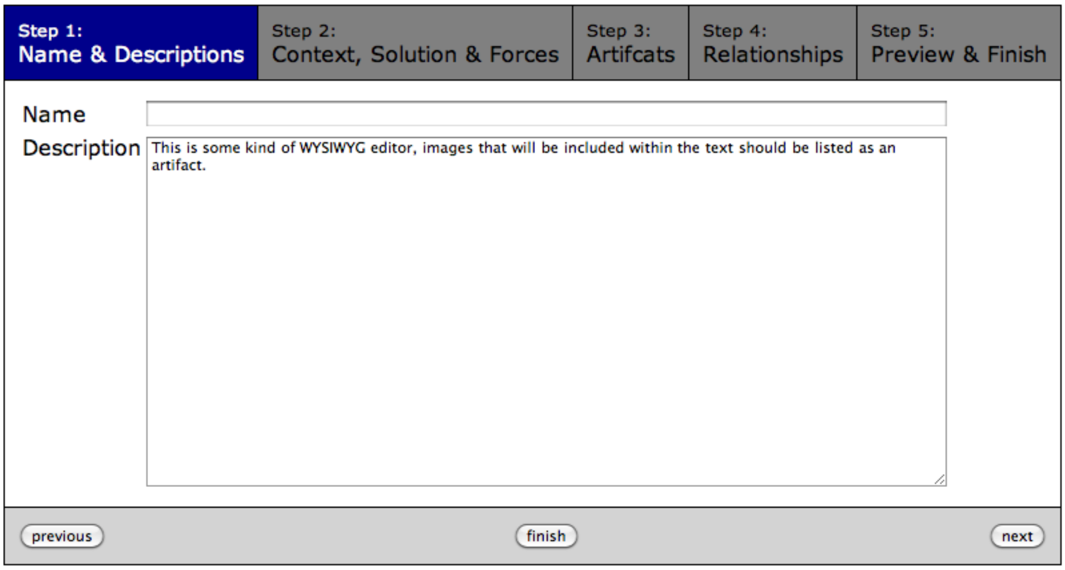
A2 Traceability matrix

	HLUC-1	HLUC-2	HLUC-3	HLUC-4	HLUC-5	HLUC-6	HLUC-7	HLUC-8	HLUC-9	HLUC-10	HLUC-11
SPUC-1.1	X										
SPUC-1.2	X										
SPUC-2.1		X									
SPUC-2.2		X									
SPUC-2.3		X									
SPUC-3.1			X								
SPUC-3.2			X								
SPUC-4.1				X							
SPUC-4.2				X							
SPUC-4.3				X							
SPUC-5.1					X						
SPUC-6.1						X					
SPUC-6.2						X					
SPUC-6.3						X					
SPUC-6.4						X					
SPUC-7.1							X				
SPUC-8.1								X			
SPUC-9.1									X		
SPUC-9.2									X		
SPUC-9.3									X		
SPUC-10.1										X	
SPUC-10.2										X	
SPUC-10.3										X	
SPUC-11.1											X
SPUC-11.2											X
SPUC-11.3											X

A3 Category tree



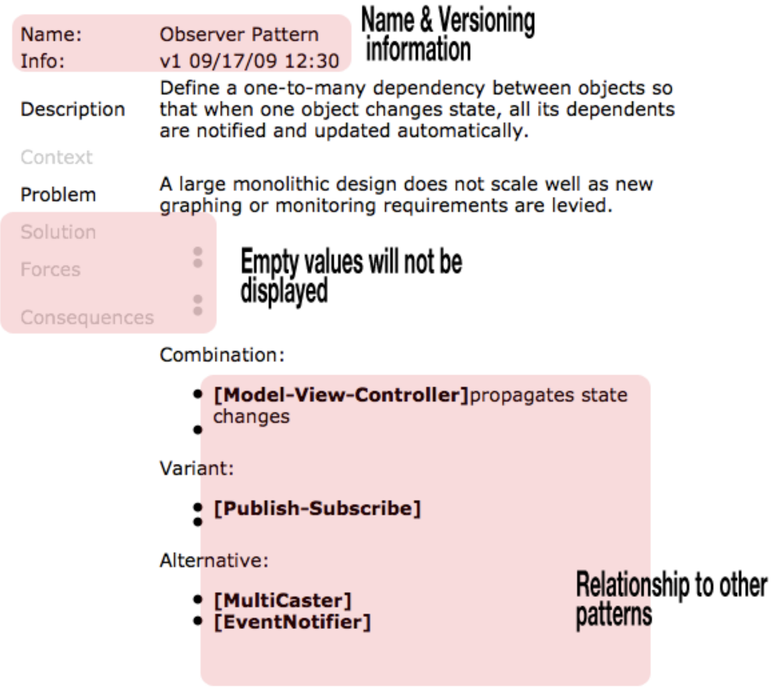
A4 Outlines



Name

Description This is some kind of WYSIWYG editor, images that will be included within the text should be listed as an artifact.

Figure 6: Add Pattern



Name: Observer Pattern
Info: v1 09/17/09 12:30

Name & Versioning information

Description Define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically.

Context

Problem A large monolithic design does not scale well as new graphing or monitoring requirements are levied.

Solution

Forces

Consequences

Empty values will not be displayed

Combination:

- **[Model-View-Controller]**propagates state changes
-

Variant:

- **[Publish-Subscribe]**
-

Alternative:

- **[MultiCaster]**
- **[EventNotifier]**

Relationship to other patterns

Figure 7: View Pattern

A5 Usability Measurement Matrix

Req.	Instruments	Unit	Current	Worst	Planned	Best	Observed
SUPR-Q1	Observation	points	0	75	90	100	-
SUPR-Q2							
SUPR-Q3							
SUPR-Q4							
SUPR-Q5							
SUPR-Q6							
SUPR-Q7							