Visual Code Review

Group 5

Daniel Andersson Tenninge

Gustaf Carleson

Johan Björk

Patrik McKiernan

1. Introduction

Document	Design Document
System	Visual Code Review
Intended readership	Project members, developers and stakeholders
Date	2008-03-08

Version history

Version	Rationale and changes	Date	Authors
1.0	First version	2008-03-08	Johan Björk Daniel Tenninge Gustaf Carleson Patrik McKiernan

1.1. Overview

This is the Design document for the VCR system provide the design details of the system.
This document is intended for the developers of the VCR system and will work as a guideline in the process of developing and implementing the system. Stakeholders in this project will also have an interest in this document, mostly due to the section describing the user interface.
This document explains the fundamental architectural design of this system, and will work as a foundation for the implementation phase. It outlines the software architecture by structuring different major components into a hierarchical layer with different levels of detail.
With this document in hand, it will be possible for developers to implement a working model of the intended system.

1.2. Abstract

This is the Design document of the VCR system, a system that will significantly aid in the peer code reviewing policy that many software companies employ. The code review process states that before a developer can incorporate his piece of code into the whole system, it must be reviewed by a fellow developer to ensure that good software quality will be guaranteed in the project.

In several software companies the code review process is still a process that is severely overlooked. When a developer wants someone to review his code he needs to find someone, in person, that actually has the time to review his code at the moment. This will obstruct him from starting to write new code or he will have to disturb someone in the middle of his or her work. This system will take care of handling these connections between developers and their respective reviewers. By making it possible for reviewers to review code when they actually have time for it, will free up time for developers and reviewers alike, allowing them to continue with their work.

1.3. References

NR	Document
1	Requirements document for the VCR system, version 1.0

1.4. Important terms

Abbreviation / term	Complete word or phrase	Description
VCR	Visual Code Review	The name of the system under development this document is referring to.
SCM	Source Code Management	A system for controlling multiple revisions in the source code development process.
SVN	Subversion	An SCM system. See http://subversion.tigris.org/ for more information.
MySQL	MySQL	A multi-threaded, multi-user SQL DBMS system.
DBMS	Database Management System	Software system for managing databases.
PySVN	PySVN	A bridge between Python scripts and Subversion system.
UML	Unified Modeling Language	An object modeling and specification language.

Innehållsförteckning

2. System Overview

2.1. General Description

Visual Code Review is a system designed to ease the code review process. Code reviewing is the process to let other developers in a team to verify the quality of the other persons' source code. We accomplish this by intercepting all source code submitted to the company's Version Control server, and presenting the source code that needs to be reviewed on a web site.

Non intrusive

The system is designed to be as little intrusive to the daily work of the developers as possible.

Automatic

The system automatically intercepts source code that matches a certain policy, and puts them up on the web interface. It further also automatically forwards the source code to the main company repository when the review is completed.

Mail

The system uses mail to notify developers the status on the review status and comments made to their source code that is under review.

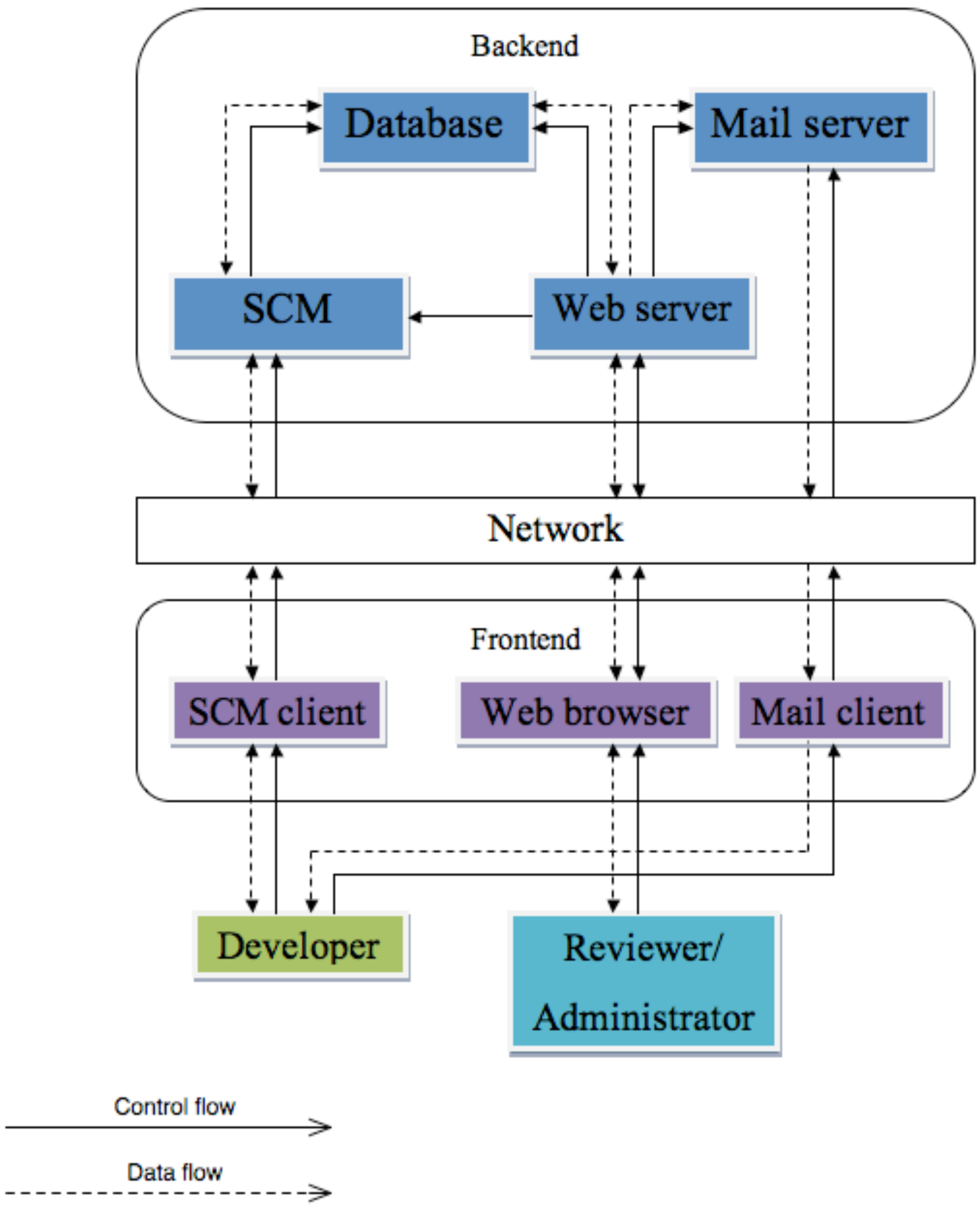
Subversion

The system supports the Subversion Review Control System to intercept source code that needs to be reviewed, and present it on the frontend.

Design

Our system is composed of two parts, the backend and the frontend. The backend collects the source code to be presented for review, inserting it to a common database. The front end allows the users to verify and leave comments on the source code in a simple manner.

2.2. Overall Architecture Description



The reviewer or administrator uses a standard web browser to connect to the back-end web server part of the VCR system. The user is presented with information in a graphical user interface. Through the connection with the web server the reviewer is presented with the current commits and can perform code reviews from the browser. The administrator is instead presented with possibilities of administrating reviewers and policies through the web interface.

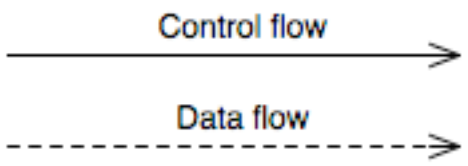
The web server is a HTTP web server backed up by a PHP interpreter to generate dynamic HTML pages for the web clients. The web server has a connection to the database, through which it can list the commits, with their source code, and information about reviewers and policies. Through PHP the web server has a connection to the mail server, which it uses to send mails to the developer about the updated status on their commits e.g. whether the commit got accepted or not.

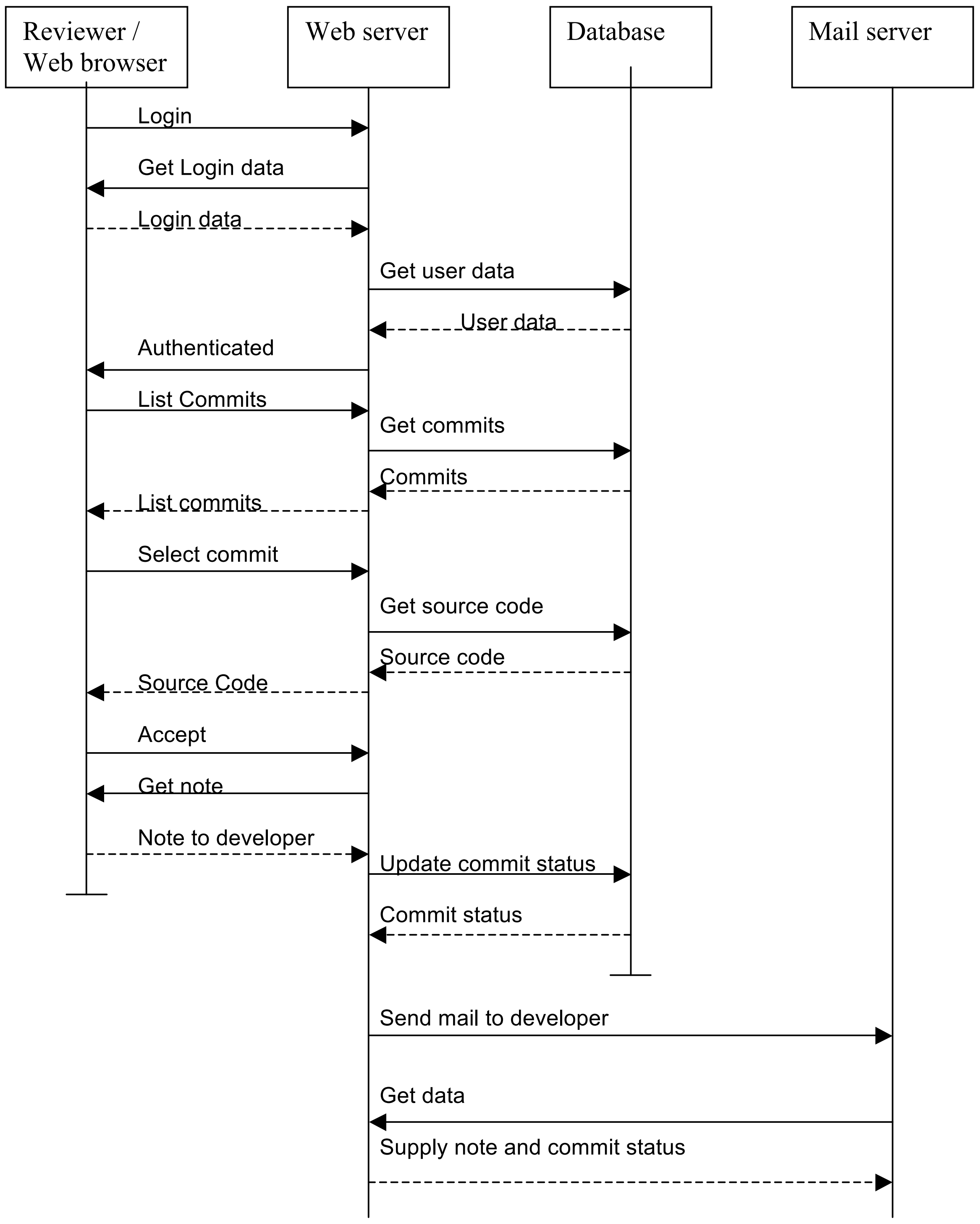
The developer uses a Source Code Management (SCM) client to send commits to the SCM server. The server is notified of the commit and runs python scripts, which in turn checks the commit and the branch it belongs with what policy it belongs to.
If it needs to be reviewed then the script stores the commit in the database, awaiting review. The code is otherwise directly committed to the SCM system’s storage. It also notifies the developer of the whether or not the commit did need reviewing.

2.3. Detailed Description

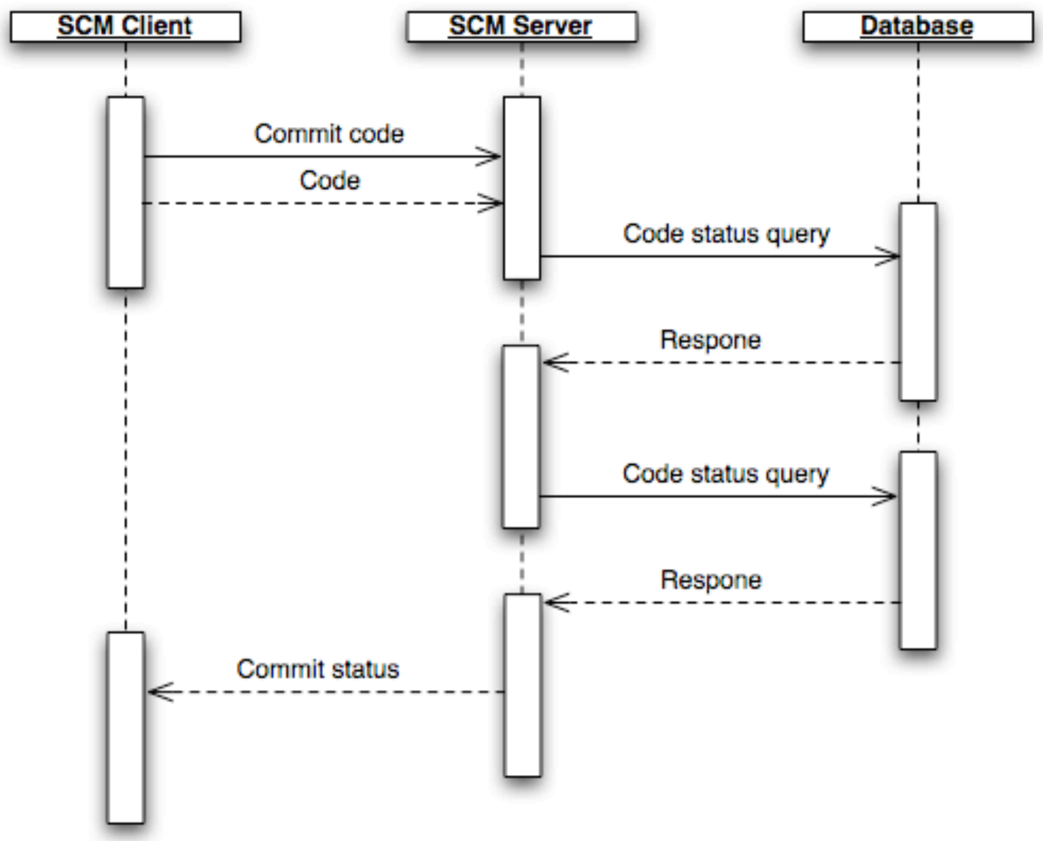
2.3.1. Review procedure

Sequence diagrams of examples of communication between components in the system as defined in the Unified Modeling Language (UML).





2.3.2. Commit process

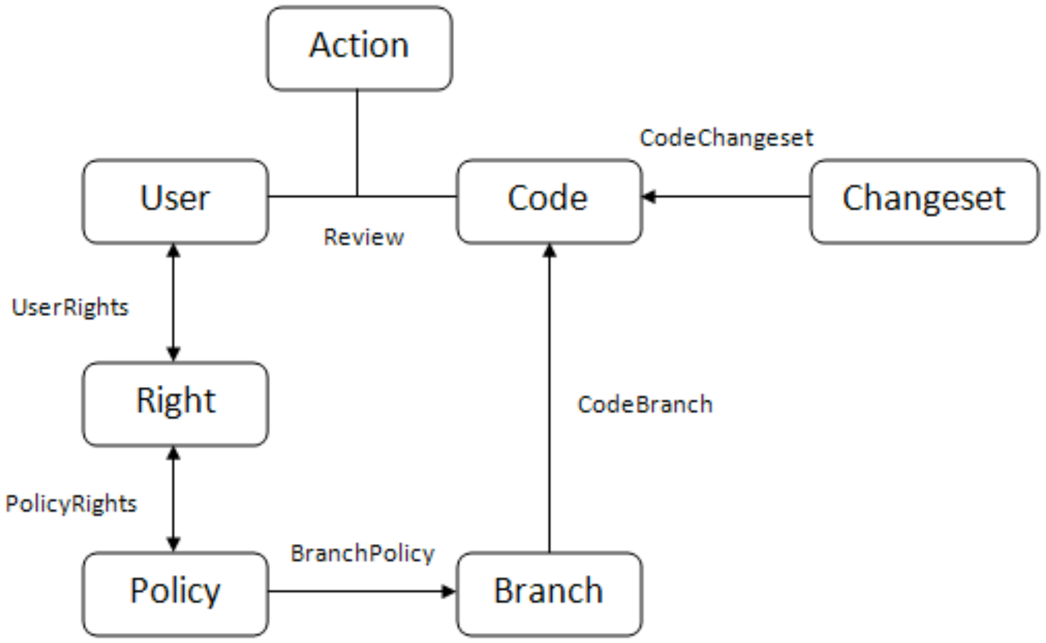


2.3.3. Database diagram

This is an entity/relationship diagram describing the database used by the system.

Relationship descriptions:

—	1:1
→	1:n
↔	m:n



Entity / Relationship	***Description***
User	Contains user information such as login, password and e-mail.
Right	Describes the review rights a user can posess (i.e. C++, SQL, req. engineering).
Action	The actions that a reviewer can take (i.e. accept, reject).
Policy	Consists of a set of rights to be applied to a branch.
Code	Contains the committed code and its relevant information.
Branch	Represents a branch in the Source Control Management (SCM) development.
Changeset	Every set of code committed belongs to a changeset.
Review	Connects objects which are part of a review.
UserRights	Connects users with their rights.
PolicyRights	Connects policies with their rights.
BranchPolicy	Connects a policy to one or more branches.
CodeBranch	Connects a piece of code to a branch.
CodeChangeset	Connects a piece of code to a changeset.

3. Design Considerations

3.1. Assumptions and Dependencies

3.1.1. Related software and hardware

The back end as well as the front end of the system requires several external programs to be installed on the server / servers.

- A SQL database engine. We will use and support MySQL.
- A web server with PHP support. We will use and support Apache.
- A source code management tool. We will use and support SVN.
- Pygments
- PySVN
- Mail server supported by PHP.

3.1.2. Possible and/or probable changes in functionality

Future updates may contain an extended support of source code management tools as well as extended policy rules.

3.2. General Constraints

Generally speaking, our system is constrained mostly by third party software. These are specifically the;

- Web server, in how many users it can serve pages to effectively.
- SCM system, in how fast it handles commit operations.
- PHP interpreter, in how quickly it interprets web logic.
- Database system, in how rapidly it stores and retrieves information.
- MTA (Mail Transfer Agent).

Thus, all of these systems constraints apply to our system. Some of these are server CPU (Central Processing Unit) performance, HDD (Hard Disk Drive) space, amount of RAM (Random Access Memory) installed, other third party software needed, system time precision etc.

The availability of our system is also a constraint since it does not function if one or more of the third party systems are not available. This can be countered with known techniques such as back-up servers, load balancing, multiple network connections, UPS's (Uninterruptible Power Supply) etc.

The Requirements Document (Sec. 4 Non-functional requirements) also presents a few system constraints concerning the availability and the ease of use of the system which is going to require testing.

The system scalability is not a constraint since load balancing of web traffic is possible and because the SCM system, database system, web server and MTA can reside in different computers. These have to be connected by a network which is however constraint by bandwidth (i.e. how much information can be transferred over the network at any one time) and how high the response time of the network is.

Mainly, the possible number of concurrent users and the degree of effectivity of the review and developing process is affected by the overall availability and performance of the system.

4. Graphical User Interface

4.1. Overview of the user interface

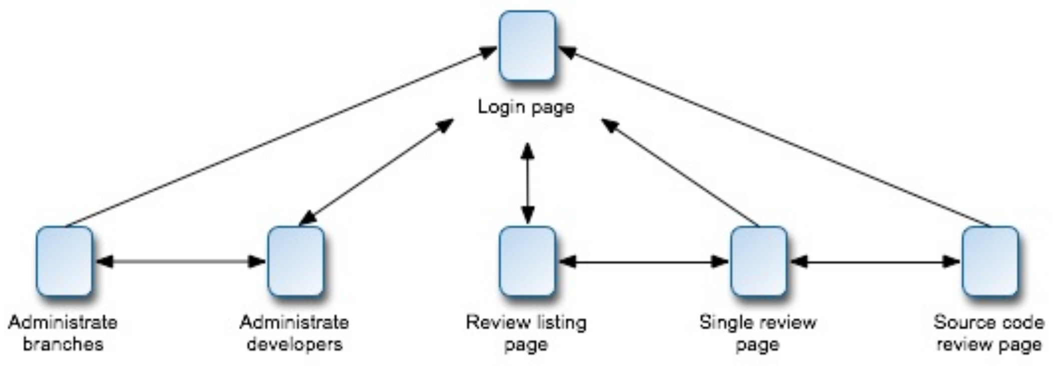


Figure 4. 1

The user is first presented with a login page. After logging in the user will be directed to either the administrator or the review part depending on the user privileges. The administrator will first be presented with the page for administrating developers. The administrator can then switch back and forth between the pages for administrating branches or developers.
The reviewer will be directed to the review-listing page where she can see all the pending commits that she is allowed to review.
After selecting a commit, she will be presented with information about that particular commit including all the files included in the commit.
By selecting a file the user will be presented with the syntax highlighted source code.
On every page there is a log out button, which will redirect the user back to the log in page. The arrows show in which direction the user can navigate through the pages.

4.2. Concept art

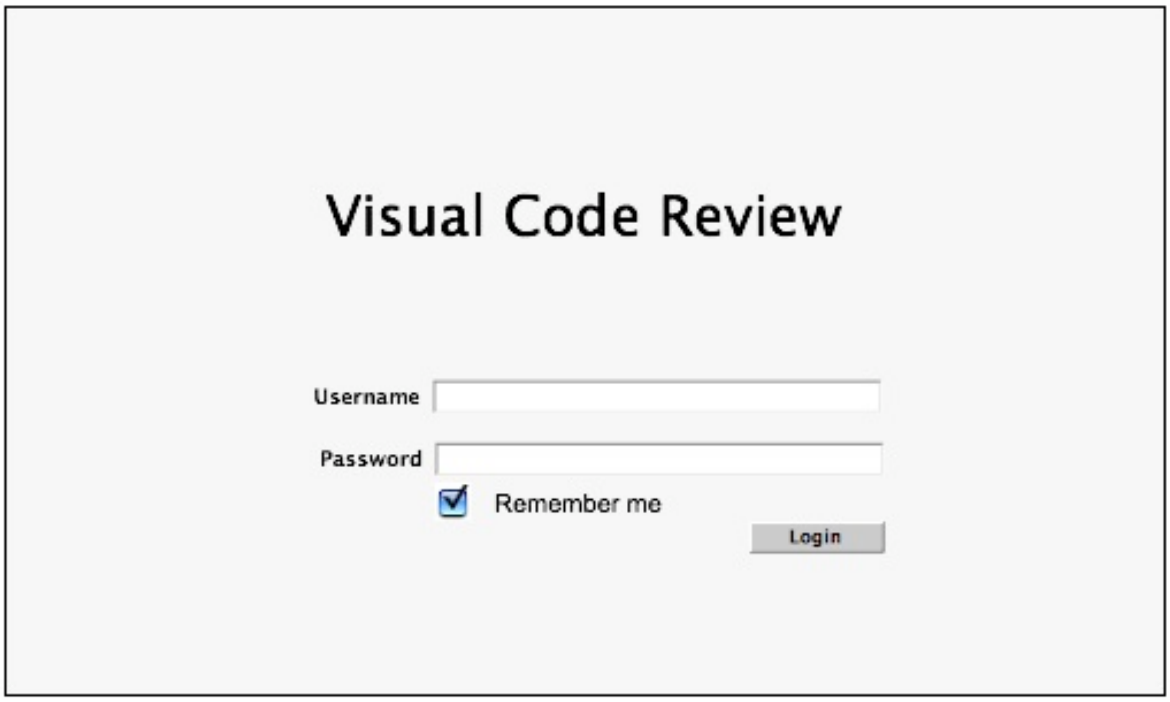


Figure 4.2.1 “Login page”

Related functional requirements
F.11

Controls and fields
txtLoginName: input for the user’s username.
txtLoginPwd: input for the user’s password.
chkRemember: checkbox used for remembering a user.
btnLogin: login button.
Triggers the event “eventAdminDev” or the “eventReview” depending on the authorization level of the user.

Directed from
The user is presented with this page, by either surfing to the web server or by the event “eventLogOut”.

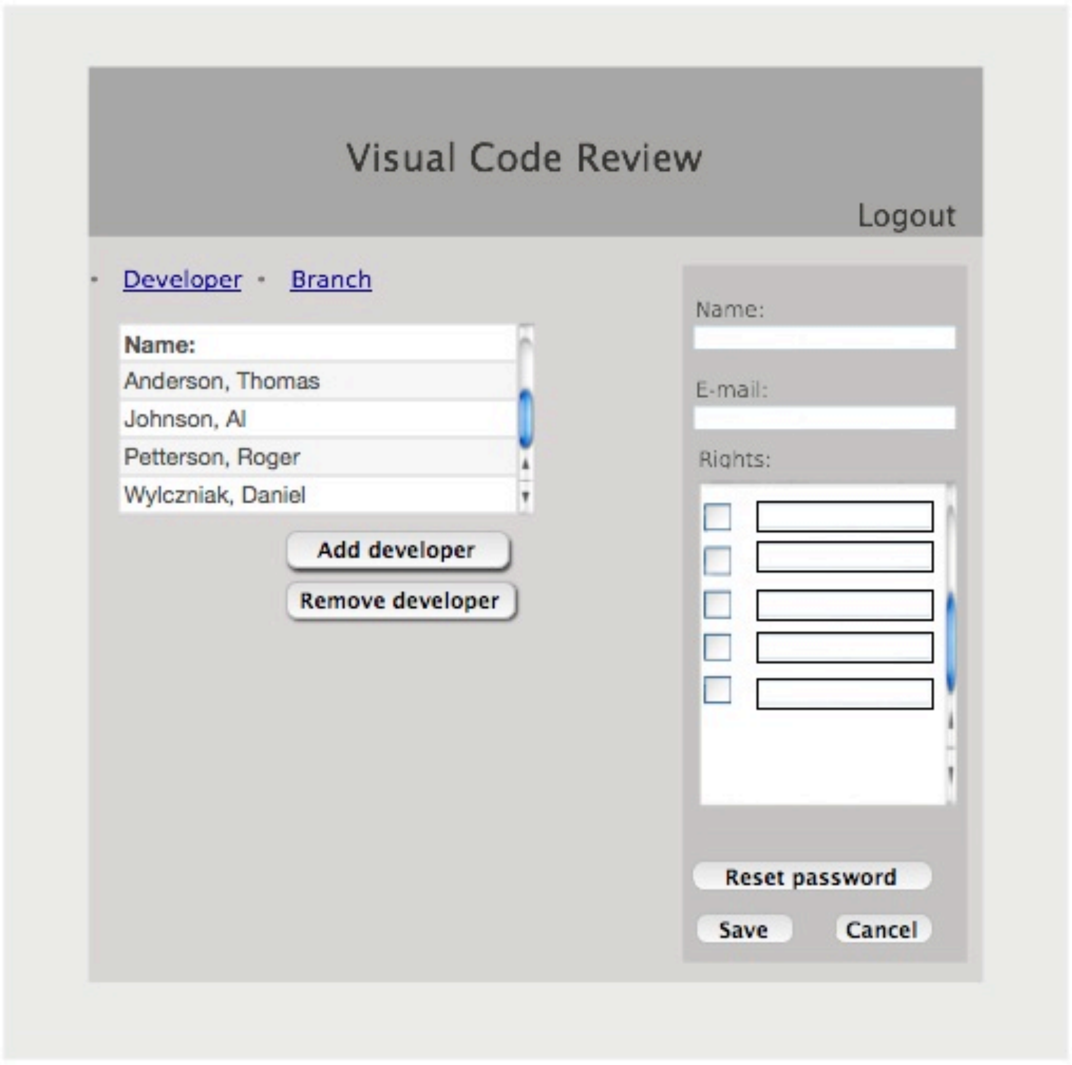


Figure 4.2.3 “Administrate developers”

Related functional requirements
F.12
F.13
F.14

Controls and fields
txtDevName: input for the developer’s name.
txtDevEmail: input for the developer’s email.
chkDevRights: input for the developer’s rights.
lnkDev: link leading to the current page.
Triggers the event “eventAdminDev”.
lnkBranch: link leading to the administration of branches page.
Triggers the event “eventAdminBranches”
btnAddDev: button used for adding a new developer.
Triggers the event “eventAddDev”.
btnRmvDev: button used for removing selected developer.
Triggers the event “eventRmvDev”.
btnResetPwd: button used for resetting a developers password.

Triggers the event “eventResetPassword”
btnSaveDev: button used for saving changes made to a developer.
Triggers the event “eventSaveDev”
btnCancelDev: button used to reset changes made to a developer.
Triggers the event “eventCancelDev”.
btnLogOut: button used to log out.
Triggers the event “eventLogOut”.

Directed from
The administrator reaches this page through the event “eventAdminDev”.

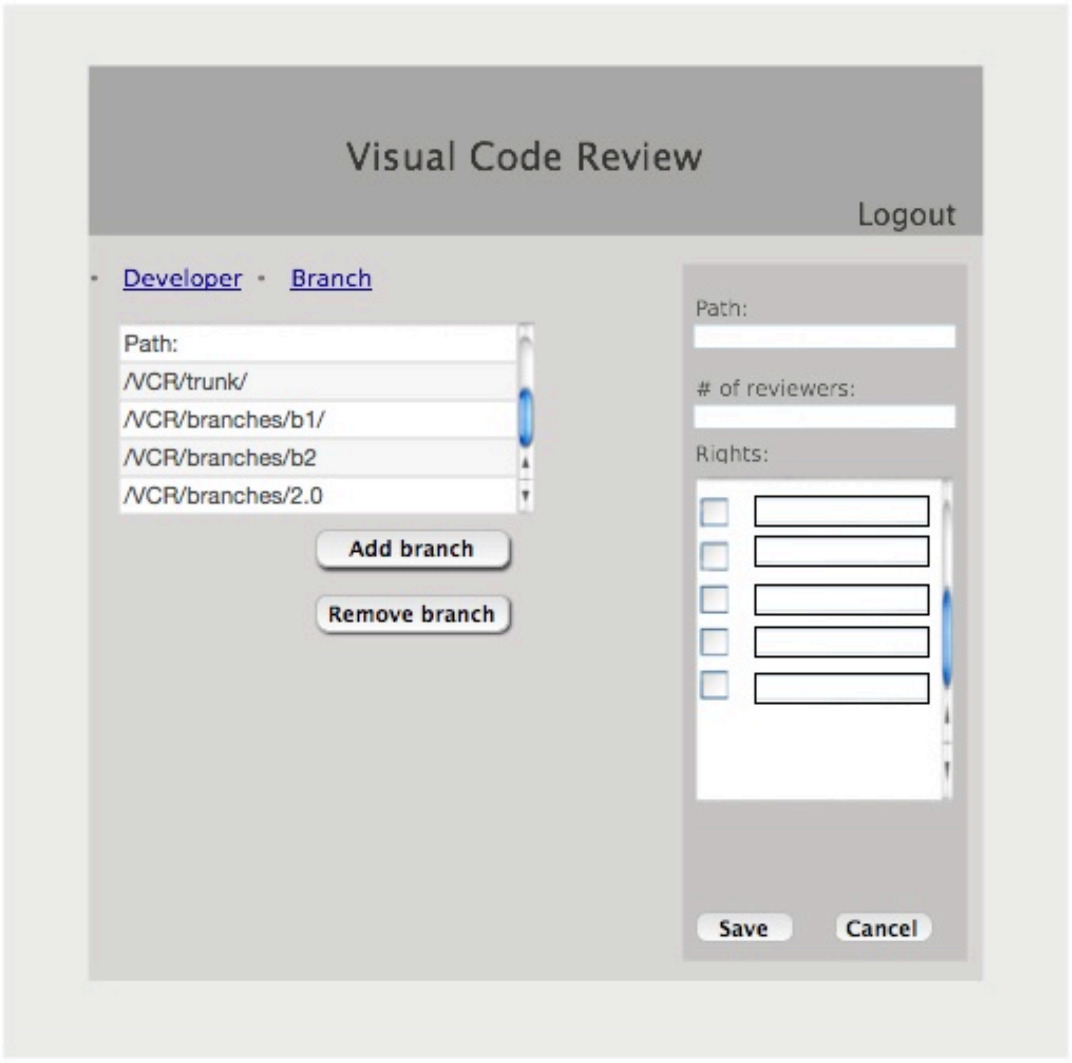


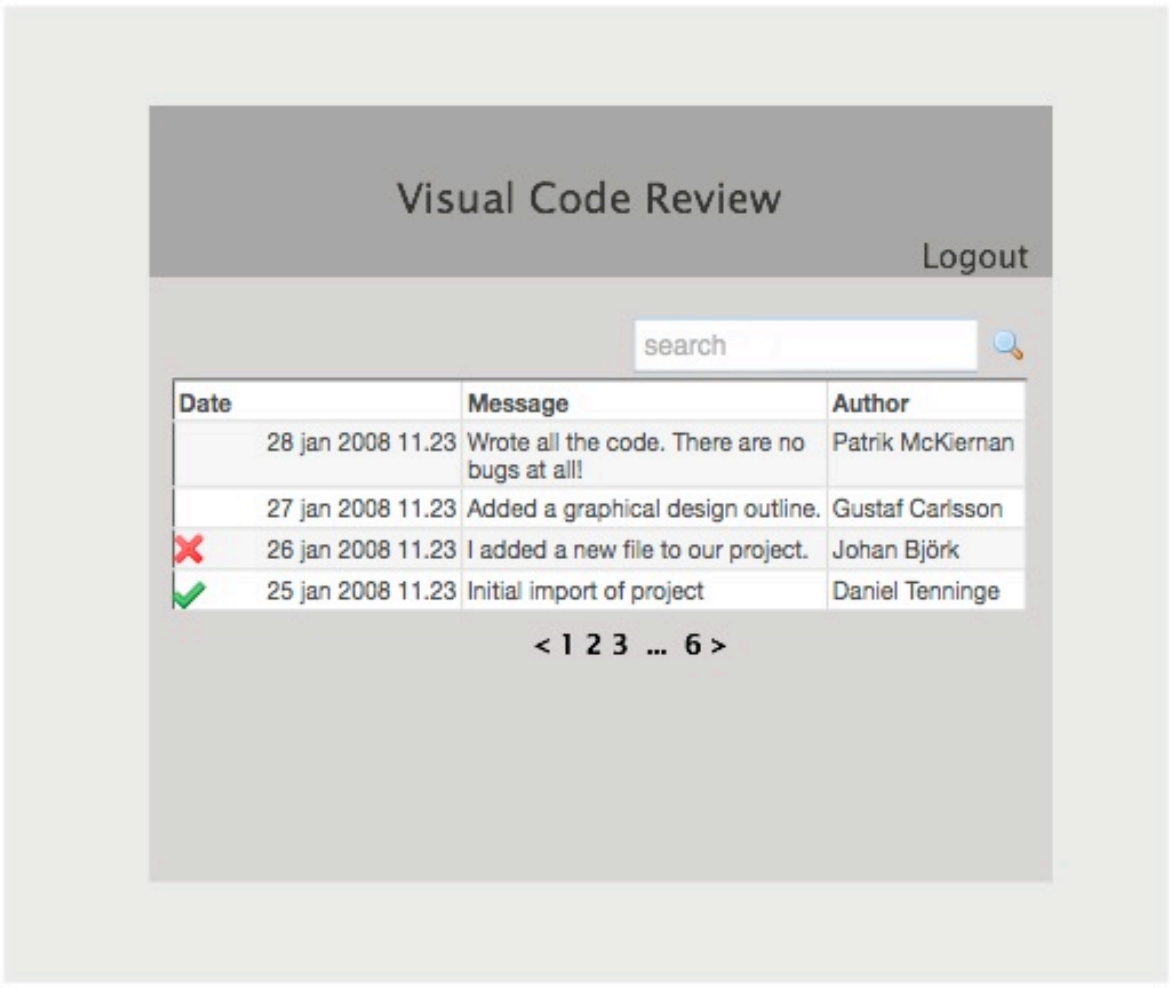
Figure 4.2.4 “Administrate Branches”

Related functional requirements
F.13
F.14

Controls and fields
txtPath: input for the branches path.
txtNrOfReviewers: input for the number of reviewers necessary. **chkBranchRights**: input for the branches rights.
InkDev: link leading to the current page.
Triggers the event “eventAdminDev”.
InkBranch: link leading to the administration of branches page.
Triggers the event “eventAdminBranches”
btnAddBranch: button used for adding a new branch.
Triggers the event “eventAddBranch”.
btnRmvBranch: button used for removing selected developer.
Triggers the event “eventRmvBranch”.
btnSaveBranch: button used for saving changes made to a branch.

Triggers the event “eventSaveBranch”
btnCancelBranch: button used to reset changes made to a branch.
Triggers the event “eventCancelBranch”.
btnLogOut: button used to log out.
Triggers the event “eventLogOut”.

Directed from
The administrator reaches this page through the event “eventAdminBranches”.



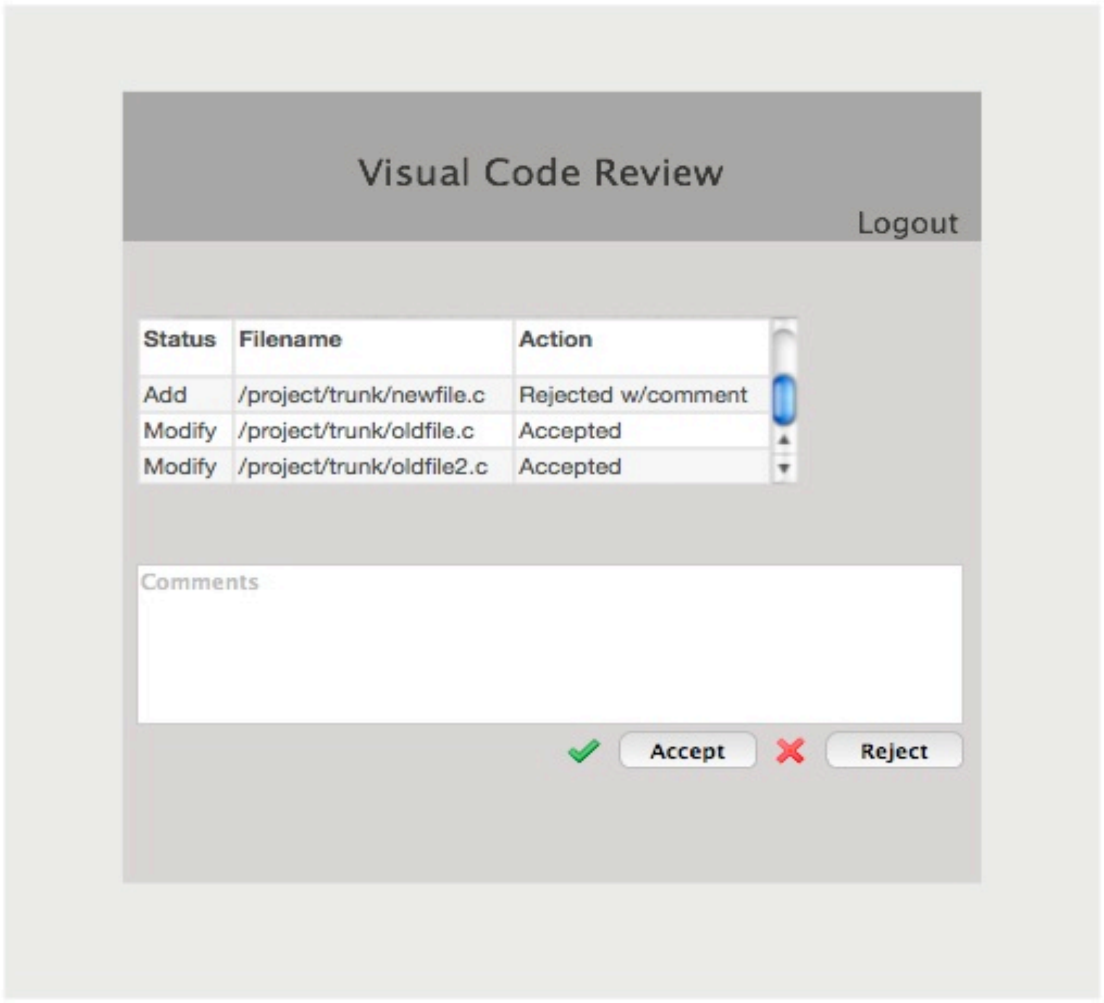
Date	Message	Author
28 jan 2008 11.23	Wrote all the code. There are no bugs at all!	Patrik McKiernan
27 jan 2008 11.23	Added a graphical design outline.	Gustaf Carlsson
✗ 26 jan 2008 11.23	I added a new file to our project.	Johan Björk
✓ 25 jan 2008 11.23	Initial import of project	Daniel Tenninge

Figure 4.2.4 “Review overview page”

Related functional requirements
F.5

Controls and fields
txtSearchField: specify search criteria.
btnSearch: button used for searching on the criteria.
Triggers the event “eventSearch”.
btnSelectPage: button used to select a review listing page.
Triggers the event “eventSelectPage”.
btnLogOut: button used to log out.
Triggers the event “eventLogOut”.
itmMenuSelect: menu item used to show details about a commit.
Triggers the event “eventListReview”.

Directed from
The developer reaches this page from the event “eventReview”.



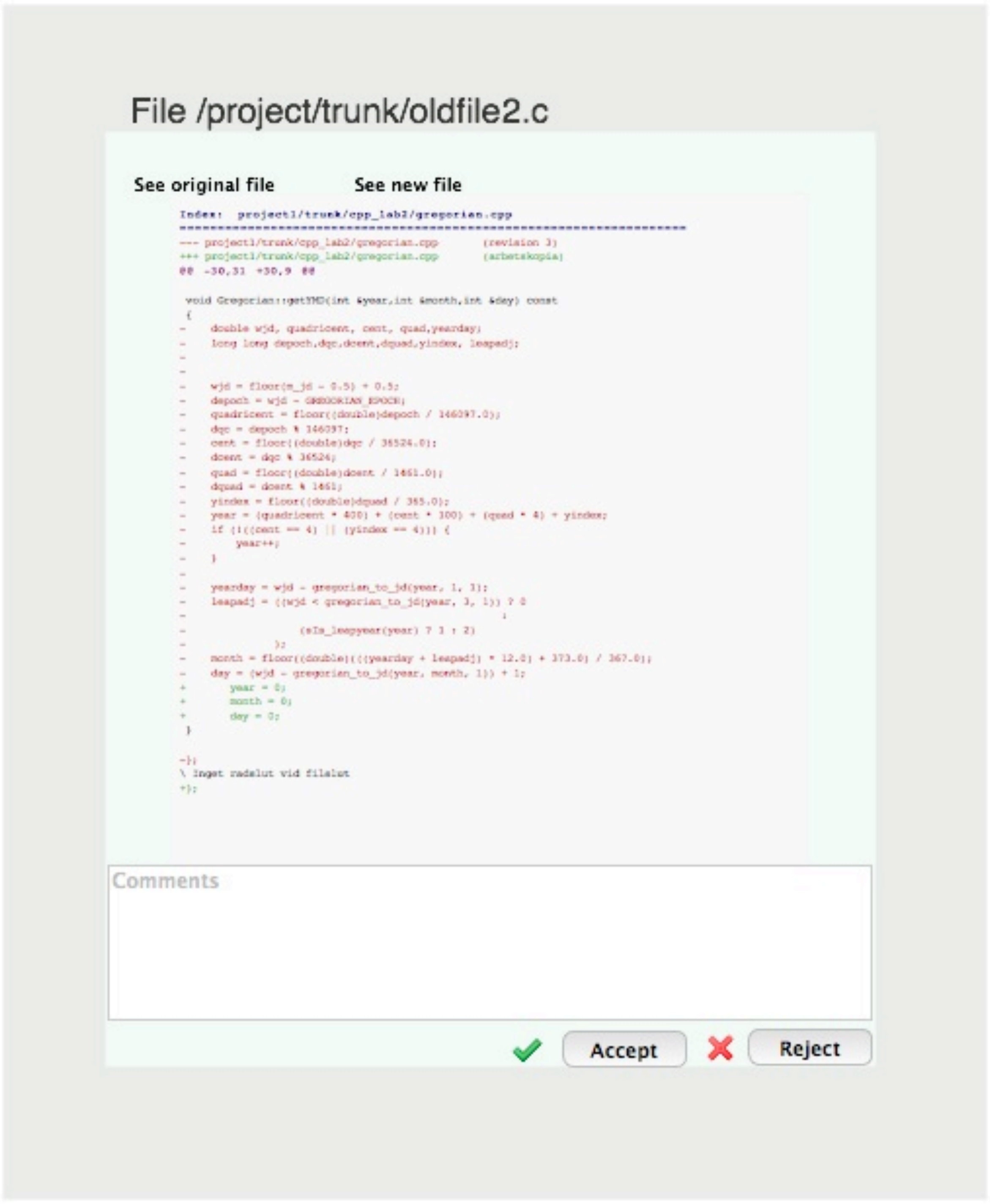
Status	Filename	Action
Add	/project/trunk/newfile.c	Rejected w/comment
Modify	/project/trunk/oldfile.c	Accepted
Modify	/project/trunk/oldfile2.c	Accepted

Figure 4.2.5 “Commit overview page”

Related functional requirements
F.5
F.6
F.7
F.8
F.10

Controls and fields
txtComments: Reviewer can input a comment to the displayed changeset.
btnChangeSetAccept: The reviewer can accept the changeset.
Triggers the event “eventAcceptChangeset”
btnChangeSetReject: The reviewer can reject the changeset.
Triggers the event ”eventRejectChangeset”
btnLogOut: button used to log out.
Triggers the event “eventLogOut”.
itmMenuSelectFile: menu item used to show details about a file.
Triggers the event “eventListFile”.

Directed from
The developer reaches this page from the event “eventListReview”.



```
Index:  project1/trunk/cpp_lab2/gregorian.cpp
===================================================================
--- project1/trunk/cpp_lab2/gregorian.cpp       (revision 3)
+++ project1/trunk/cpp_lab2/gregorian.cpp       (arbetskopia)
@@ -30,31 +30,9 @@

 void Gregorian::getYMD(int &year,int &month,int &day) const
 {
-    double wjd, quadricent, cent, quad,yearday;
-    long long depoch,dqc,dcent,dquad,yindex, leapadj;
-
-
-    wjd = floor(m_jd - 0.5) + 0.5;
-    depoch = wjd - GREGORIAN_EPOCH;
-    quadricent = floor((double)depoch / 146097.0);
-    dqc = depoch % 146097;
-    cent = floor((double)dqc / 36524.0);
-    dcent = dqc % 36524;
-    quad = floor((double)dcent / 1461.0);
-    dquad = dcent % 1461;
-    yindex = floor((double)dquad / 365.0);
-    year = (quadricent * 400) + (cent * 100) + (quad * 4) + yindex;
-    if (!((cent == 4) || (yindex == 4))) {
-        year++;
-    }
-
-    yearday = wjd - gregorian_to_jd(year, 1, 1);
-    leapadj = ((wjd < gregorian_to_jd(year, 3, 1)) ? 0
-                                                      :
-                  (sIs_leapyear(year) ? 1 : 2)
-              );
-    month = floor((double)(((yearday + leapadj) * 12.0) + 373.0) / 367.0);
-    day = (wjd - gregorian_to_jd(year, month, 1)) + 1;
+       year = 0;
+       month = 0;
+       day = 0;
 }

-};
\ Inget radslut vid filslut
+};
```

Figure 4.2.6 “Source code page”

Related functional requirements
F.5
F.6
F.7

Controls and fields
lnkOriginalFile: used to display the original file.
Triggers the event “eventDisplayOldFile”.

lnkNewFile: used to display the new file.
Triggers the event “eventDisplayNewFile”.
txtComments: Reviewer can input a comment to the displayed sourcecode.
btnAcceptFile: button used to accept this file.
Triggers the event “eventAcceptFile”.
btnRejectFile: button used to reject this file.
Triggers the event “eventRejectFile”.

Directed from
The developer reaches this page from the event “eventListFile”.

5. Design detail

5.1. Class Responsibility Collaborator (CRC) Cards

5.1.1. VCR System Classes

Class CheckCommit	
Responsibilities	**Collaborators**
Intercept commit Check if a commit needs reviewing. Insert information and source code into the database.	BEDatabase Highlight Transaction

Class BEDatabase	
Responsibilities	**Collaborators**
Handle connection to the database for the backend of the system.	CheckCommit.

Class Highlight	
Responsibilities	**Collaborators**
Handle connection to the syntax highlighting engine.	CheckCommit. Pygments

Class FEDatabase	
Responsibilities	**Collaborators**
Handle the connection to the database, for the frontend of the system.	Standard

Class Standard	
Responsibilities	**Collaborators**
Provide a standard web page template. Redirect to login page if user is not authenticated.	FEDatabase AdminPage Review

Class AdminPage	
Responsibilities	**Collaborators**
Provide a standard web page template for AdminDev and AdminBranch.	Standard AdminDev AdminBranch

Class AdminDev	
Responsibilities	**Collaborators**
Administrate developers	AdminPage AdminBranch

Class AdminBranch	
Responsibilities	**Collaborators**
Administrate branches	AdminPage AdminDev

Class Review	
Responsibilities	**Collaborators**
List commits available for review. List information about a single commit. Show the source code from a single file in the commit.	Standard

5.1.2. Third party classes

Class Pygments	
Provided by: Pygments.org v. 0.9	
Responsibilities	**Collaborators**
Syntax highlights source code.	Highlight

Class Transaction	
Provided by: Pysvn.tigris.org v. 1.5.2	
Responsibilities	**Collaborators**
Extract information from SVN	CheckCommit

5.2. Class diagram

The class diagrams are represented using Unified Modeling Language (UML).

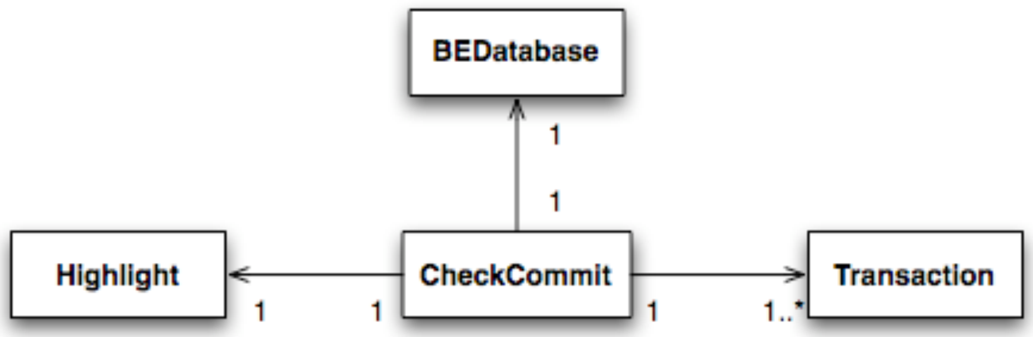


Figure 5.2.1: Backend

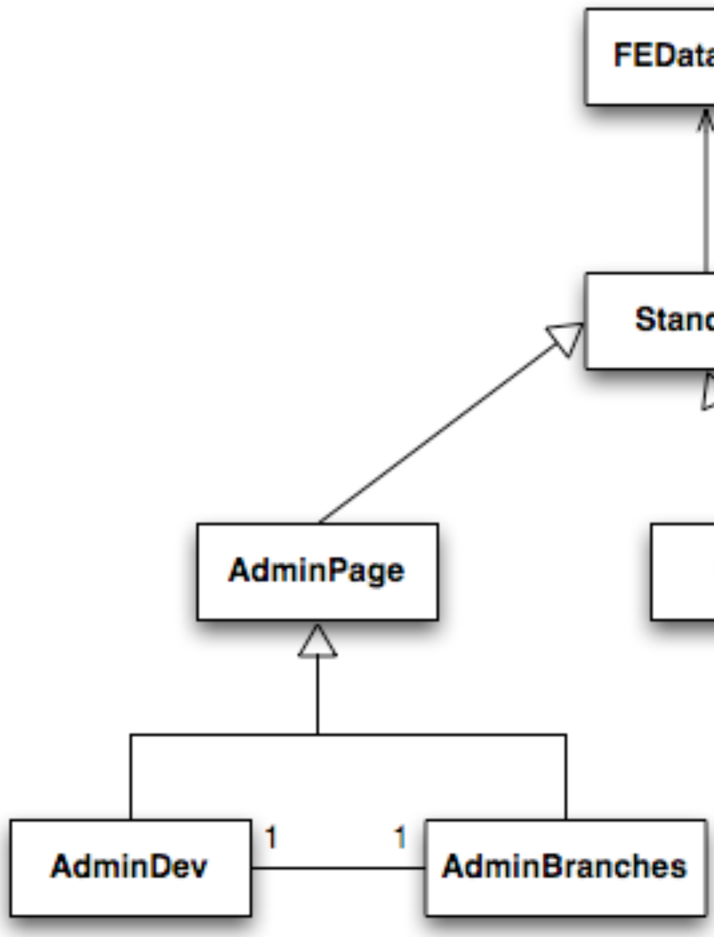


Figure 5.2.2: Frontend

5.3. State Chart

The state charts are represented using Unified Modeling Language (UML).

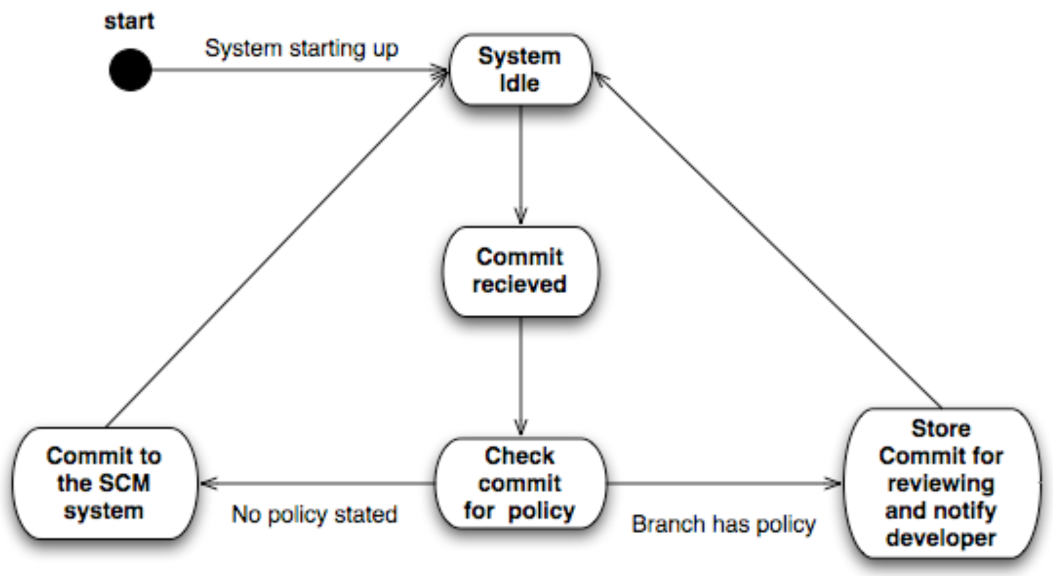


Figure 5.3.1: Commit interception

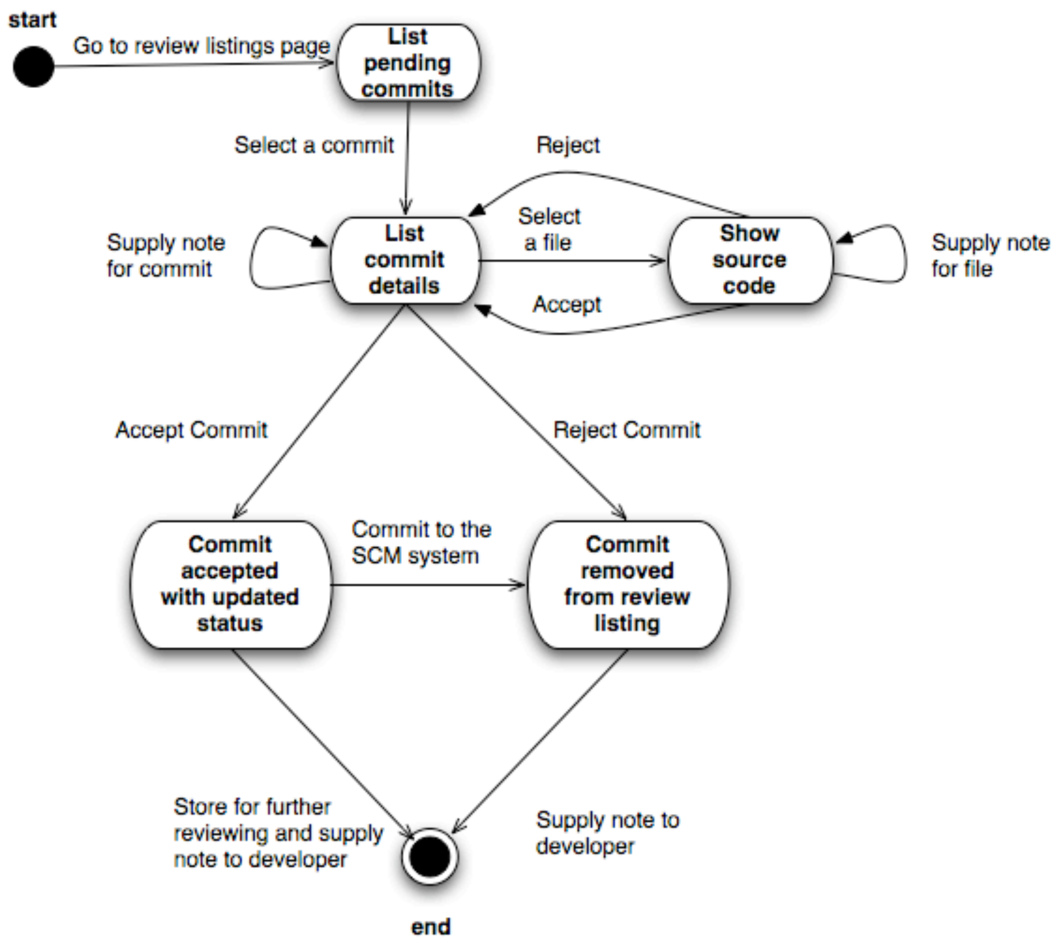


Figure 5.3.2: Review process

5.4. Interaction diagram

The sequence interaction diagrams are represented using Unified Modeling Language (UML).

5.4.1. Developer Commit

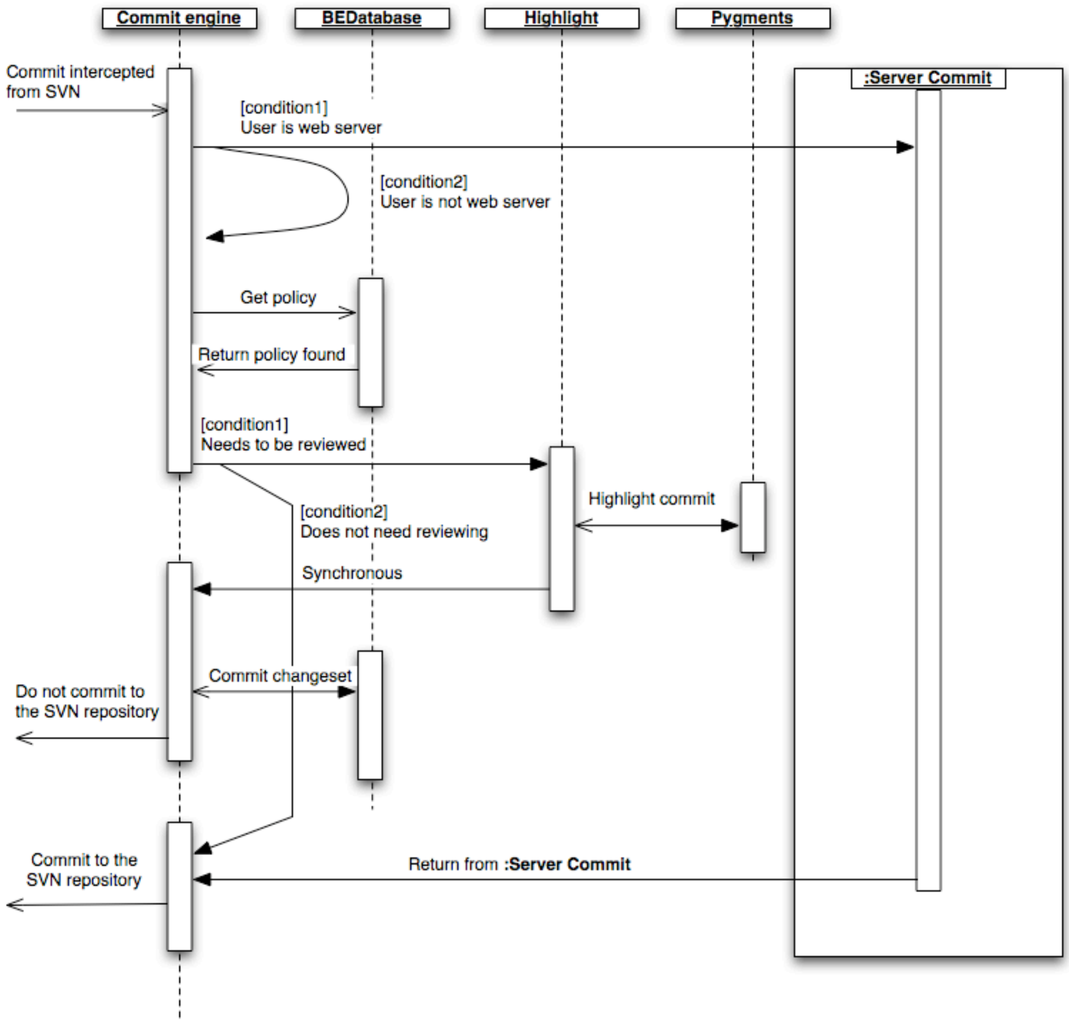

Commit engine
BEDatabase
Highlight
Pygments
:Server Commit
Commit intercepted from SVN
[condition1] User is web server
[condition2] User is not web server
Get policy
Return policy found
[condition1] Needs to be reviewed
[condition2] Does not need reviewing
Highlight commit
Synchronous
Commit changeset
Do not commit to the SVN repository
Commit to the SVN repository
Return from :Server Commit

Figure 5.4.1.1

5.4.2. Server commit

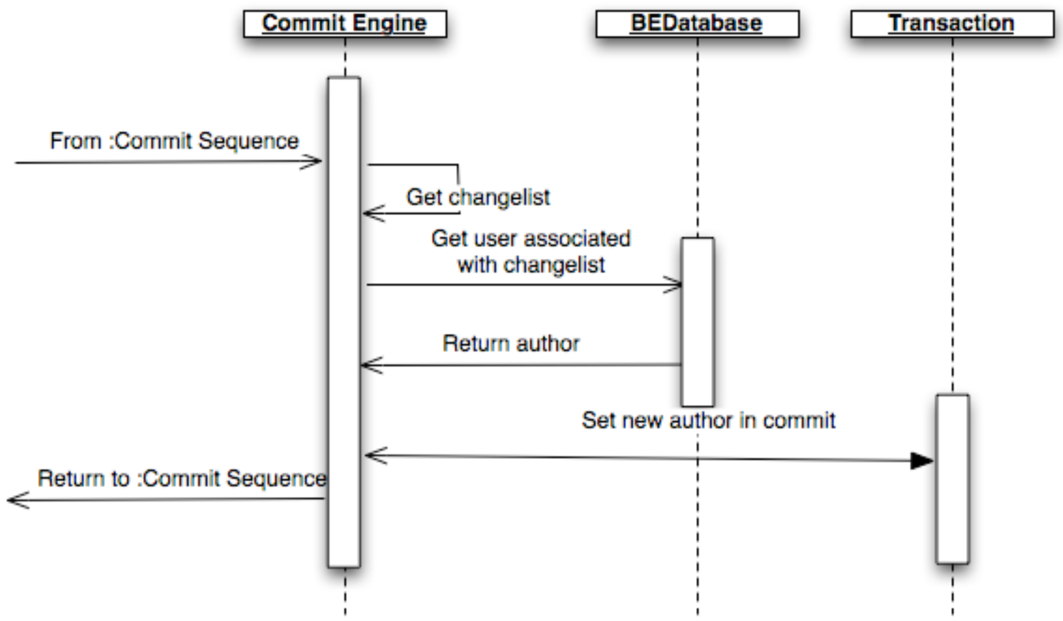


Figure 5.4.2.1

5.4.3. Review process

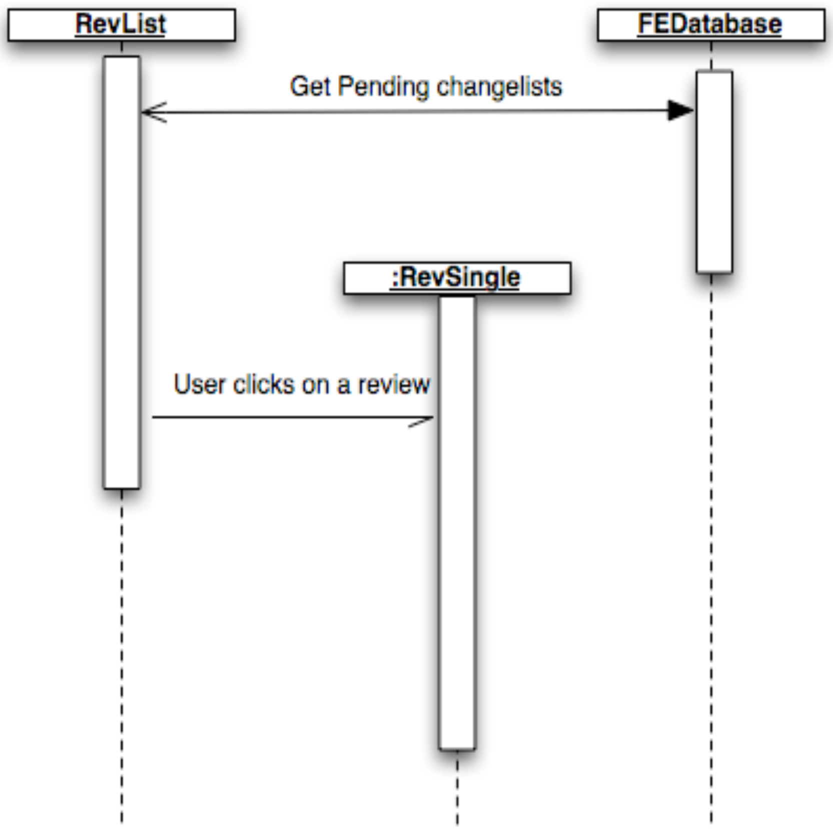


Figure 5.4.3.1

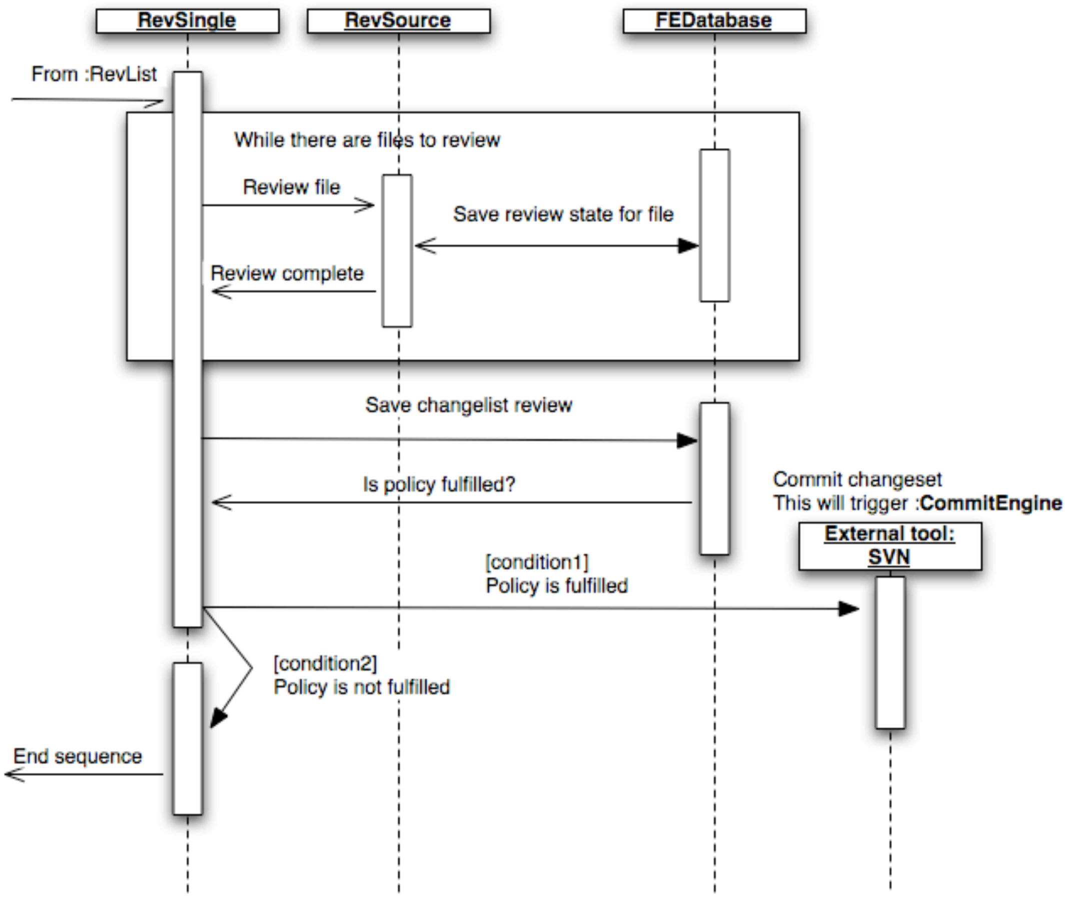

RevSingle
RevSource
FEDatabase
From :RevList
While there are files to review
Review file
Save review state for file
Review complete
Save changelist review
Is policy fulfilled?
Commit changeset
This will trigger :CommitEngine
External tool: SVN
[condition1]
Policy is fulfilled
[condition2]
Policy is not fulfilled
End sequence

Figure 5.4.3.2

5.5. Detailed Design

5.5.1. Classes

Class Name:	CheckCommit
Attributes:	db: A open connection to the database
Methods:	Main
Related Functional Requirement:	F.1, F.2

Method Name:	main()
Parameters:	External: SVN transaction id, SVN repository path.
Return Value:	Nonzero: Commit shall be aborted Zero: Commit shall proceed.
Description:	Inserts a changeset into the database if the code is affected by a review policy, otherwise commits to SVN server.
Data Structures:	n/a.
Pre-conditions:	n/a.
Validity Checks:	n/a.
Post-conditions:	Commit has been processed.
Called by:	External: SVN pre-commit hook i.e. a blackbox provided by SVN.
Calls:	BEDatabase, Highlight External: PySVN

Class Name:	Highlight
Attributes:	None
Methods:	highlightDiff()
Related Functional Requirement:	n/a

Method Name:	highlightDiff ()
Parameters:	Filename,oldfile(contents),newfile(contents)
Return Value:	Highlighted diff
Description:	Syntax highlights the diff between two files in HTML format.
Data Structures:	n/a.
Pre-conditions:	n/a.
Validity Checks:	n/a.
Post-conditions:	Syntax is highlighted in HTML
Called by:	CheckCommit
Calls:	External: Pygments

Class Name:	BEDatabase
Attributes:	db: A open connection to the database

Methods:	__init__
	importSQL()
	doSQL()
	newChangeSet()
	newCode()
	getPolicy()
	getAuthor()

Related Functional Requirement:	F.3

Method Name:	__init__()
Parameters:	Host, user, password, database,
Return Value:	n/a. Throws exception
Description:	Imports a .sql file
Data Structures:	db; Connection to database.
Pre-conditions:	n/a.
Validity Checks:	That we can connect to the database.
Post-conditions:	Connected to the database.
Called by:	CheckCommit
Calls:	External: Python mysql module.

Method Name:	importSQL()
Parameters:	Filename
Return Value:	n/a. Throws exception
Description:	Imports a .sql file
Data Structures:	n/a.
Pre-conditions:	n/a.
Validity Checks:	n/a
Post-conditions:	File has been executed by SQL engine.
Called by:	None
Calls:	doSQL

Method Name:	doSQL()
Parameters:	statement,arguments,cursor
Return Value:	Number of rows and row data. Depending on cursor selected. Throws exception
Description:	Executes statement, replacing any occurances of % with a argument from the argumentlist. (sql injection

	safe)
Data Structures:	n/a.
Pre-conditions:	n/a.
Validity Checks:	n/a.
Post-conditions:	SQL statement is executed.
Called by:	BEDatabase.
Calls:	External: Python mysql module.

Method Name:	newChangeSet()
Parameters:	log, author, date
Return Value:	New changeset ID.
Description:	Creates a new changeset.
Data Structures:	n/a.
Pre-conditions:	n/a.
Validity Checks:	n/a.
Post-conditions:	New changeset created
Called by:	CheckCommit
Calls:	doSQL

Method Name:	newCode()
Parameters:	changeSetId,filename,newcontent,hldiff
Return Value:	n/a.
Description:	Creates a new code entry associated with changeSetId.
Data Structures:	n/a.
Pre-conditions:	changeSetId exists in the database.
Validity Checks:	n/a
Post-conditions:	New code is associated with the changeSetId.
Called by:	CheckCommit
Calls:	doSQL

Method Name:	getPolicy(filename)
Parameters:	Filename
Return Value:	policyID.
Description:	Searches for an active policy that the filename is affected by. If more then one matching is available, it will return a valid matching, which one is not defined.
Data Structures:	n/a.
Pre-conditions:	n/a.
Validity Checks:	n/a.
Post-conditions:	A policyid is returned if available.
Called by:	CheckCommit
Calls:	doSQL

Method Name:	getAuthor(changeSetId)
Parameters:	changesetID
Return Value:	Author of that changeset.
Description:	Returns the svn-author of the changeset.
Data Structures:	n/a.
Pre-conditions:	The changeset is valid.
Validity Checks:	n/a
Post-conditions:	Author is returned.
Called by:	CheckCommit
Calls:	doSQL

Class Name:	Standard
Attributes:	

Methods:	__construct()
	check_auth()
	check_login()
	get_login()
	logout()

Related Functional Requirement:	F.11

Method Name:	__construct()
Parameters:	n/a
Return Value:	n/a
Description:	Constructor for the class Standard.
Data Structures:	n/a
Pre-conditions:	n/a
Validity Checks:	n/a
Post-conditions:	n/a
Called by:	n/a
Calls:	check_auth()

Method Name:	check_auth
Parameters:	n/a
Return Value:	Boolean, true if authenticated false otherwise.
Description:	Checks if the user is logged in.
Data Structures:	n/a
Pre-conditions:	n/a

Validity Checks:	n/a
Post-conditions:	n/a
Called by:	__construct()
Calls:	n/a

Method Name:	get_login()
Parameters:	n/a
Return Value:	n/a
Description:	Redirect the user to the login page.
Data Structures:	n/a
Pre-conditions:	n/a
Validity Checks:	n/a
Post-conditions:	n/a
Called by:	check_auth()
Calls:	n/a

Method Name:	logout()
Parameters:	n/a
Return Value:	n/a
Description:	Ends the users session.
Data Structures:	n/a
Pre-conditions:	The user is logged in.
Validity Checks:	n/a
Post-conditions:	n/a
Called by:	Standard
Calls:	n/a

Class Name:	AdminPage
Attributes:	n/a

Methods:	link_admindev()
	link_adminbra()

Related Functional Requirement:	n/a

Method Name:	link_admindev()
Parameters:	n/a
Return Value:	Link to the admin developer page.
Description:	Returns a link to the admin developer page.
Data Structures:	n/a
Pre-conditions:	Logged in with admin rights.
Validity Checks:	n/a
Post-conditions:	n/a

Called by:	AdminDev, AdminBranch
Calls:	n/a

Method Name:	link_adminbra()
Parameters:	n/a
Return Value:	Link to the admin branches page.
Description:	Returns a link to the admin branches page.
Data Structures:	n/a
Pre-conditions:	Logged in with admin rights.
Validity Checks:	n/a
Post-conditions:	n/a
Called by:	AdminDev, AdminBranch
Calls:	n/a

Class Name:	AdminDev
Attributes:	n/a

Methods:	add_developer()
	remove_developer()
	update_developer()
	reset_pwd()

Related Functional Requirement:	F.12, F.13, F.14

Method Name:	add_developer()
Parameters:	n/a
Return Value:	False if adding failed, true otherwise.
Description:	Adds a new developer to the database.
Data Structures:	USR_User.
Pre-conditions:	Logged in as administrator.
Validity Checks:	n/a
Post-conditions:	A new developer is added to the database.
Called by:	eventAddDev
Calls:	n/a

Method Name:	remove_developer()
Parameters:	developer_id
Return Value:	False if remove failed, true otherwise.
Description:	Remove a developer from the database.
Data Structures:	USR_User.

Pre-conditions:	Logged in as administrator.
Validity Checks:	Check that a developer is selected, show alert dialog if none is selected.
Post-conditions:	A developer is removed from the database.
Called by:	eventRemoveDev
Calls:	n/a

Method Name:	update_developer()
Parameters:	developer_id developer_name developer_email developer_rights
Return Value:	True if updated was successful, false otherwise.
Description:	Updates information about a developer.
Data Structures:	USR_User.
Pre-conditions:	Logged in as administrator.
Validity Checks:	Check that all fields are non-empty, if not so show an alert dialog.
Post-conditions:	Developer is updated with new data.
Called by:	eventSaveDev.
Calls:	n/a

Method Name:	reset_pwd()
Parameters:	developer_id
Return Value:	False if reset failed, true otherwise.
Description:	Reset a developers password.
Data Structures:	USR_User.
Pre-conditions:	Logged in as administrator.
Validity Checks:	Check that a developer is selected, show alert dialog if none is selected.
Post-conditions:	A developers password is reset.
Called by:	eventResetPassword
Calls:	n/a

Class Name:	AdminBranches
Attributes:	n/a

Methods:	add_branch()
	remove_branch()
	update_branch()

Related	F.13, F.14

Functional Requirement:	

Method Name:	add_branch()
Parameters:	n/a
Return Value:	False if adding failed, true otherwise.
Description:	Adds a new branch to the database.
Data Structures:	BRA_Branch
Pre-conditions:	Logged in as administrator.
Validity Checks:	n/a
Post-conditions:	A new branch is added to the database.
Called by:	eventAddBranch
Calls:	n/a

Method Name:	remove_branch()
Parameters:	branch_id
Return Value:	False if the remove failed, true otherwise.
Description:	Removes a branch from the database.
Data Structures:	BRA_Branch
Pre-conditions:	Logged in as administrator.
Validity Checks:	Check that a branch is selected, show alert dialog if none is selected.
Post-conditions:	A branch is removed from the database.
Called by:	eventRemoveBranch
Calls:	n/a

Method Name:	update_branch()
Parameters:	branch_id branch_path nr_of_reviewers rights
Return Value:	True is update was successful, false otherwise.
Description:	Updates
Data Structures:	Updates information about a branch.
Pre-conditions:	Logged in as administrator.
Validity Checks:	Check that all the fields are non-empty and show an alert dialog if not.
Post-conditions:	The branch is updated with the new data.
Called by:	eventSaveBranch
Calls:	n/a

Class Name:	Review
Attributes:	n/a

Methods:	get_reviews()
	get_review_details()
	reject_commit()
	accept_commit()
	get_source_code()
	reject_source()
	accept_source()
	send_note()

Related Functional Requirement:	F.4, F.5, F.6, F.7, F.8, F.9, F.10

Method Name:	get_reviews()
Parameters:	search_string page_nr nr_per_page
Return Value:	Data for the specified page.
Description:	Retrieves a list containing all commits up for review for the specified page based on the developers rights.
Data Structures:	CHS_Changeset
Pre-conditions:	Logged in as developer.
Validity Checks:	n/a
Post-conditions:	n/a
Called by:	eventReview eventSelectPage eventSearch
Calls:	n/a

Method Name:	get_review_details()
Parameters:	commit_id
Return Value:	Returns the details of a commit.
Description:	Gets the details of a specific commit.
Data Structures:	CHS_Changeset
Pre-conditions:	Logged in as developer.
Validity Checks:	n/a
Post-conditions:	n/a
Called by:	eventListReview
Calls:	n/a

Method Name:	reject_commit()

Parameters:	commit_id note_string
Return Value:	True if reject was successful, false otherwise.
Description:	Rejects a commit and supplies note.
Data Structures:	CHS_Changeset REV_Review
Pre-conditions:	Logged in as developer.
Validity Checks:	Checks that a note is supplied, otherwise displays an alert dialog.
Post-conditions:	The specified commits is rejected.
Called by:	eventRejectChangeset
Calls:	send_note()

Method Name:	accept_commit()
Parameters:	commit_id note_string
Return Value:	True if accept was successful, false otherwise.
Description:	Accepts a commit and supplies note.
Data Structures:	CHS_Changeset REV_Review
Pre-conditions:	Logged in as developer and no source file is rejected.
Validity Checks:	Checks that a note is supplied, otherwise displays an alert dialog.
Post-conditions:	The specified commits is accepted.
Called by:	eventAcceptChangeset
Calls:	send_note()

Method Name:	get_source_code()
Parameters:	source_code_id
Return Value:	Highlighted source code.
Description:	Returns the highlighted source code for the specified file.
Data Structures:	COD_Code
Pre-conditions:	Logged in as developer.
Validity Checks:	n/a
Post-conditions:	n/a
Called by:	eventListFile
Calls:	n/a

Method Name:	reject_source_code()
Parameters:	source_code_id note_string
Return Value:	True if reject was successful, false otherwise.
Description:	Rejects a source file in the commit and supplies note.

Data Structures:	COD_Code REV_Source_Review
Pre-conditions:	Logged in as developer.
Validity Checks:	Checks that a note is supplied, otherwise displays an alert dialog.
Post-conditions:	The specified source file is rejected.
Called by:	eventRejectFile
Calls:	n/a

Method Name:	accept_source_code()
Parameters:	source_code_id note_string
Return Value:	True if accept was successful, false otherwise.
Description:	Accepts a source file in the commit and supplies a note.
Data Structures:	COD_Code REV_Source_Review
Pre-conditions:	Logged in as developer.
Validity Checks:	Checks that a note is supplied, otherwise displays an alert dialog.
Post-conditions:	The specified source file is accepted.
Called by:	eventAcceptFile
Calls:	n/a

Method Name:	send_note()
Parameters:	message email address
Return Value:	True if mail was sent, false otherwise.
Description:	Sends an email to the address specified with the supplied note.
Data Structures:	n/a
Pre-conditions:	Logged in as developer.
Validity Checks:	n/a
Post-conditions:	Email is sent to the address with supplied message.

Called by:	accept_commit()
	reject_commit()

Calls:	n/a

5.5.2. Database structure

This database that is used by both the FE and the BE-Database classes has this structure.

Table name	Fields	Type	Foreign keys
USR_User	USR_Id	INT, PRIMARY KEY	N/A
	USR_Login	VARCHAR(10)	N/A
	USR_Password	TEXT	N/A
	USR_Name	VARCHAR(50)	N/A
	USR_Email	VARCHAR(50)	N/A

Table name	Fields	Type	Foreign keys
RGT_Right	RGT_Id	INT, PRIMARY KEY	N/A
	RGT_Name	VARCHAR(50)	N/A

Table name	Fields	Type	Foreign keys
POL_Policy	POL_Id	INT, PRIMARY KEY	N/A
	POL_Name	VARCHAR(50)	N/A
	POL_Category	VARCHAR(50)	N/A
	POL_AcceptedThreshold	INT	N/A
			N/A

Table name	Fields	Type	References foreign key
CHS_Changeset	CHS_Id	INT, PRIMARY KEY	N/A
	CHS_USR_Id	INT	N/A
	CHS_Comment	VARCHAR(1024)	USR_User(USR_Id)
	CHS_Date	DATE	N/A

Table name	Fields	Type	Foreign keys
BRA_Branch	BRA_Id	INT, PRIMARY KEY	N/A
	BRA_Name	VARCHAR(50)	N/A
	BRA_Path	VARCHAR(255)	N/A
	BRA_POL_Id	INT	POL_Policy(POL_Id)
	BRA_Active	BOOLEAN	N/A

Table name	Fields	Type	Foreign keys
COD_Code	COD_Id	INT, PRIMARY KEY	N/A
	COD_Path	VARCHAR(255)	N/A
	COD_Filename	VARCHAR(255)	N/A
	COD_CodeText	TEXT	N/A
	COD_Diff	TEXT	N/A
	COD_BRA_Id	INT	BRA_Branch(BRA_Id)
	COD_CHS_Id	INT	CHS_Changeset(CHS_Id)

Table name	Fields	Type	Foreign keys

RES_ReviewSource	RES_COD_Id	INT, PRIMARY KEY	COD_Code(COD_Id)
	RES_USR_Id	INT, PRIMARY KEY	USR_User(USR_Id)
	RES_Date	DATE	N/A
	RES_Comment	VARCHAR(1024)	N/A
	RES_Action	ENUM('Accept','Reject'), PRIMARY KEY	N/A

Table name	Fields	Type	Foreign keys
REC_ReviewChangeset	REC_CHS_Id	INT, PRIMARY KEY	COD_Code(COD_Id)
	REC_USR_Id	INT, PRIMARY KEY	USR_User(USR_Id)
	REC_Date	DATE	N/A
	REC_Comment	VARCHAR(1024)	N/A
	REC_Action	ENUM('Accept','Reject'), PRIMARY KEY	N/A

Table name	Fields	Type	Foreign keys
URI_UserRights	URI_USR_Id	INT, PRIMARY KEY	USR_User(USR_Id)
	URI_RGT_Id	INT, PRIMARY KEY	RGT_Right(RGT_Id)

Table name	Fields	Type	Foreign keys
POR_PolicyRights	POR_POL_Id	INT, PRIMARY KEY	POL_Policy(POL_Id)
	POR_RGT_Id	INT, PRIMARY KEY	RGT_Right(RGT_Id)

5.6. Package diagram

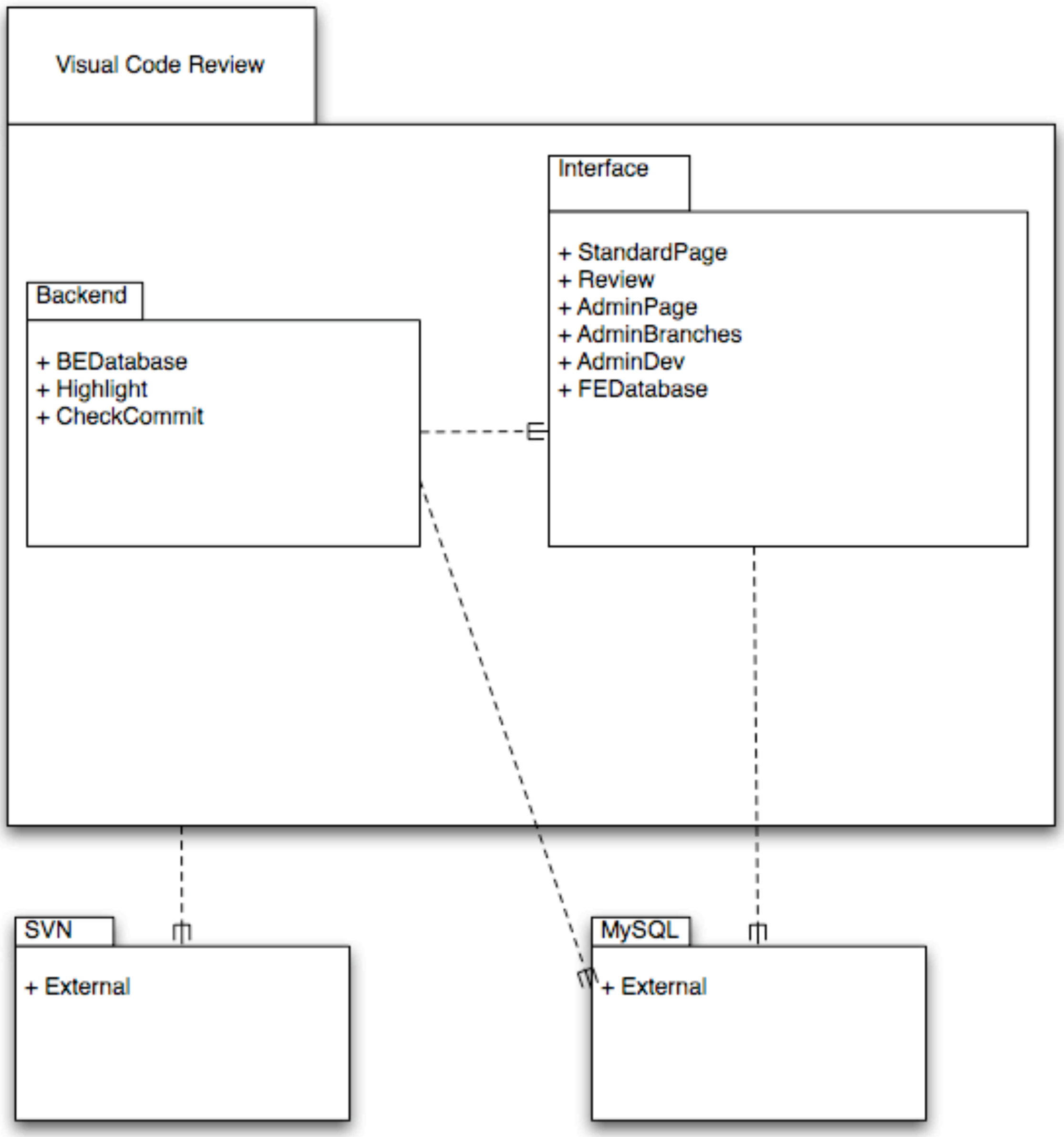
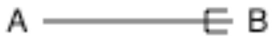


Figure 5.6.1



This means that A uses B.

6. Functional Test Cases

6.1. Login as developer

Function being tested:
Logging in to the system as a developer.

Functional Requirements:
F.11

Input:
Username and Password valid for the system.

Expected output:
The user is logged in as a developer and redirected to the Review listing page.

Instructions:
1. Write the username in the username textbox.
2. Write the password in the password textbox.
3. Press the login button.

6.2. Login as administrator

Function being tested:
Logging in to the system as an administrator.

Functional Requirements:
F.11

Input:
Username and password for a valid administrator account.

Expected output:
The user is logged in as an administrator and is redirected to the Administration page.

Instructions:
1. Write username in username textbox
2. Write password in password textbox
3. Press login

6.3. Remember me

Function being tested:
If the system can remember login data.

Functional Requirements:

F.11

Input:
Username and password for a valid account.

Expected output:
The user can automatically access any login protected page, without entering username and password again.

Instructions:
1. Write username in username textbox.
2. Write password in password textbox.
3. Click the “Remember me” checkbox.
4. Press login.
5. Close your browser window
6. Open and try to access the website.

6.4. Wrong password

Function being tested:
If the system can report invalid password.

Functional Requirements:
F.11

Input:
Username and password that do not correspond to a valid account.

Expected output:
The user is redirected back to the login page, with an indicator that the wrong password was entered.

Instructions:
1. Write username in username textbox
2. Write password in password textbox
3. Press login

6.5. Logout

Function being tested:
That you are able to logout after having logged in as either developer or administrator.

Functional Requirements:
F.11

Input:
N/A

Expected output:
The user is redirected back to the login page, having to enter a password and username to get back to the site.

Instructions:
1. Login (Follow test “Login as Developer” or “Login as Administrator)
2. Press Logout

6.6. Add Developer

Function being tested:
Adding a developer to the system.

Functional Requirements:
F.12

Input:
N/A

Expected output:
A new developer is added to the systems database.

Instructions:
1. Click the “Add developer” button on the “Administrate developers” page.

6.7. Remove Developer

Function being tested:
Remove a developer to the system.

Functional Requirements:
F.12

Input:
N/A

Expected output:
A developer is removed from the systems database.

Instructions:
1. Click the “Remove developer” button on the “Administrate developers” page.

6.8. Save new changes

Function being tested:
Save new changes made to a selected developer.

Functional Requirements:
F.12

Input:
Name and email for the developer and any selected rights for that developer.

Expected output:
The information is about the developer is updated and is shown when she is selected from the list of developers.

Instructions:
1. Enter the “Administrate developers” page.
2. Select a developer.
3. Make changes to the information displayed.
4. Click the “Save” button.

6.9. Cancel changes

Function being tested:
Cancellation of changes made to a developer.

Functional Requirements:
F.12

Input:
N/A

Expected output:
The information displayed about the developer is set back to what is stored, thereby removing any changes made.

Instructions:
1. Enter the “Administrate developers” page.
2. Select a developer.
3. Make changes to information.
4. Click the “Cancel button”.

6.10. Reset password

Function being tested:
Resetting a developer’s password.

Functional Requirements:
F.12

Input:
N/A

Expected output:
A new randomized password is set and is then email to the selected developer.

Instructions:
1. Select a developer.
2. Click on the “Reset password” button on the “Administrate developers” page.

6.11. Test save with no selected developer

Function being tested:
Trying to click the save button with no selected developer.

Functional Requirements:
F.12

Input:
N/A

Expected output:
An alert dialog shall be displayed informing the user that no developer is selected and therefore the save function is not applicable.

Instructions:
1. Enter the “Administrate developer” page.
2. Click on the “Save” button.

6.12. Switch to branch

Function being tested:
Switching from administrating developers to administrating branches.

Functional Requirements:
F.14

Input:
N/A

Expected output:
The administrator is redirected from the “Administrate developers” page to the “Administrate branches” page.

Instructions:
1. Enter the “Administrate developers” page.
2. Click on the link titled “Branches”.

6.13. Select developer

Function being tested:
Selecting a particular developer from the list of developers.

Functional Requirements:
F.12

Input:
N/A

Expected output:
A developer is selected and information about the developer is displayed in the fields. The information displayed includes, the developers name and email and also the rights of the developer.

Instructions:
1. Enter the “Administrate developers” page.
2. Select a developer from the list of developers by clicking on a name.

6.14. Add branch

Function being tested:
Adding a branch and setting policies to it.

Functional Requirements:
F.14

Input:
Path to where the branch is located, number of reviewers required and their rights.

Expected output:
The new branch gets added to the list of available branches.

Instructions:
1. Press the “Add branch” button.
2. Enter the path to the branch in the path textbox.
3. Enter number of reviewers required in the # of reviewers’ textbox.
4. Select the rights required by the reviewers.
5. Press the save button.

6.15. Remove branch

Function being tested:
Removing all the review policies from a branch.

Functional Requirements:

F.14

Input:
The branch selected for removal.

Expected output:
The selected branch is removed from the list of branches being controlled by review policies.

Instructions:
1. Select a branch from the list over available branches.
2. Press the remove branch button.

6.16. Save new changes

Function being tested:
Saving new changes to an already added branch.

Functional Requirements:
F.14

Input:
Path to where the branch is located, number of reviewers required and their rights.

Expected output:
The information about the branch is updated with the new one.

Instructions:
1. Select a branch from the list over available branches.
2. Change the information in the appropriate text boxes.
3. Press the save button.

6.17. Cancel changes

Function being tested:
Cancel an ongoing change in the information about a branch.

Functional Requirements:
F.14

Input:
None.

Expected output:
All the text boxes gets updated with the old information about the selected branch and none of the new one is stored.

Instructions:

1. Select a branch in the list over available branches.
2. Make some changes in the text boxes.
3. Press the cancel button.

6.18. Test save with no selected branch

Function being tested:
Trying to change settings without having selected a branch.

Functional Requirements:
F.14

Input:
Path to where the branch is located, number of reviewers required and their rights.

Expected output:
A dialog box will appear stating the no branch is selected. None of the new information is stored.

Instructions:
1. Make sure the no branch is selected.
2. Enter some information in the text boxes.
3. Press the save button.

6.19. Switch to developer

Function being tested:
Switching from the branch settings page to the developer settings page.

Functional Requirements:
F.12

Input:
None.

Expected output:
The user is presented with the developer settings page.

Instructions:
1. Press the developer link.

6.20. Select branch

Function being tested:
Selecting a branch.

Functional Requirements:

F.14

Input:
None.

Expected output:
The branch gets selected and its information is shown in the text boxes.

Instructions:

1. Select a branch in the list over available branches.

6.21. Search

Function being tested:
That you are able to search the changesets

Functional Requirements:
F.2
F.5

Input:
A search string

Expected output:
The user is presented with a RevList with only matching items

Instructions:

1. Follow testcase 1.1 to login
2. Enter a search string into the searchfield and press enter

6.22. Select changeset

Function being tested:
That you can select changesets

Functional Requirements:
F.2
F.5

Input:
Select a changeset

Expected output:
The user is redirected to the Review Single page for the selected changeset.

Instructions:

1. Follow testcase 1.1 to login
2. Select any changeset

6.23. Change page

Function being tested:
That you can display all pages of the changesets.

Functional Requirements:
F.2
F.5

Input:
Selected pagenumber

Expected output:
The user is presented with a Rev Listing with different items.

Instructions:
1. Follow testcase 1.1 to login
2. Select a pagenumber

6.24. Select a file in a commit

Function being tested:
To display the source code of a file in a commit.

Functional Requirements:
F.4

Input:
The selected file to display.

Expected output:
A new form showing the selected file content and a field where you can leave a comment.

Instructions:
1. In a browser window, log in as a developer.
2. Open a commit up for review.
3. Select a file by clicking at the name of the file in the list to open it.

6.25. Denial of commit acceptance

Function being tested:
The denial of acceptance of the entire commit if at least one file in it has been rejected.

Functional Requirements:
F.6

Input:

The commit or change set that you want to accept.

Expected output:
A message informing the user that part of or the entire commit has been rejected.

Instructions:
1. In a browser window, log in as a developer.
2. Open a commit and reject at least one file in it.
3. Try to accept the entire commit.

6.26. Accept a commit without supplying a comment

Function being tested:
The acceptation of a commit even if the user has chosen not to comment on the action taken.

Functional Requirements:
F.6

Input:
The commit/change set that you want to accept.

Expected output:
The status of the commit will be updated in the list of commits up for review.

Instructions:
1. In a browser window, log in as a developer.
2. Select a commit up for review.
3. Accept the commit without leaving a comment.

6.27. Accept a commit by accepting all files and leaving a comment

Function being tested:
The acceptation of a commit when the user has accepted all files separately and wishes to leave a comment on the entire commit before accepting it.

Functional Requirements:
F.6

Input:
The commit/change set that you want to accept.

Expected output:
The status of the commit will be updated in the list of commits up for review.

Instructions:

1. In a browser window, log in as a developer.
2. Open a commit and accept the files in it, one by one.
3. In the window containing the files in the commit, write a comment in the comment window.
4. Try to accept the entire commit.

6.28. Reject a commit without supplying a comment

Function being tested:
The rejection of a commit when you have not supplied a comment.

Functional Requirements:
F.6

Input:
The commit/change set that you want to reject and the comment.

Expected output:
The status of the commit will be updated in the list of commits up for review and the comment saved in the database.

Instructions:
1. In a browser window, log in as a developer.
2. Select a commit up for review.
3. Reject the commit without leaving a comment.

6.29. Reject a commit and supply a comment

Function being tested:
The rejection of a commit when you choose to supply a comment.

Functional Requirements:
F.6

Input:
The commit/change set that you want to reject and the comment.

Expected output:
The status of the commit will be updated in the list of commits up for review and the comment saved in the database.

Instructions:
1. In a browser window, log in as a developer.
2. Select a commit up for review.
3. Write a comment in the comment field.
4. Reject the commit.

6.30. Accept a single file without supplying a comment

Function being tested:
The acceptation of a commit when the user doesn't supply a comment.

Functional Requirements:
F.4
F.6

Input:
The file name of the source code you want to accept and the change set it belongs to.

Expected output:
The status of the file in the commit up for review will be updated.

Instructions:
1. In a browser window, log in as a developer.
2. Select a commit up for review.
3. Select a file belonging to that commit.
4. Accept the file.

6.31. Accept a single file and leave a comment

Function being tested:
The acceptation of a single file belonging to a commit when the user has chosen to leave a comment.

Functional Requirements:
F.4
F.6

Input:
The file name of the source code you want to accept, the change set it belongs to and the comment.

Expected output:
The status of the file in the commit up for review will be updated and the comment saved in the database.

Instructions:
1. In a browser window, log in as a developer.
2. Select a commit up for review.
3. Select a file belonging to that commit.
4. Leave a comment in the comment field.
5. Accept the file.

6.32. Reject a single file without supplying a comment

Function being tested:
The rejection of a single file when the user has not supplied a comment.

Functional Requirements:
F.4
F.6

Input:
The file name of the source code you want to accept and the change set it belongs to.

Expected output:
The status of the file in the commit up for review will be updated and the comment saved in the database.

Instructions:
1. In a browser window, log in as a developer.
2. Select a commit up for review.
3. Reject the commit without leaving a comment.

6.33. Reject a single file and leave a comment

Function being tested:
The rejection of a single file belonging to a commit when the user has chosen to leave a comment.

Functional Requirements:
F.4
F.6

Input:
The file name of the source code you want to reject, the change set it belongs to and the comment.

Expected output:
The status of the file in the commit up for review will be updated and the comment saved in the database.

Instructions:
1. In a browser window, log in as a developer.
2. Select a commit up for review.
3. Select a file belonging to that commit.
4. Leave a comment in the comment field.
5. Reject the file.

6.34. Commit to branch without policy

Function being tested:
That the code will be committed to the SVN.

Functional Requirements:
F.1
F.2

Input:
A commit created by a third-party SVN client.

Expected output:
The user receives the normal message indicating success from its SVN client.

Instructions:
1. Commit to the company “Visual Code Review” protected SVN server, following normal procedures for this action.
2. Make sure it is stored safely in the SVN repository.

6.35. Commit to branch with policy

Function being tested:
That the code will not be commited directly to the SVN repository, and that it will be added into the database.

Functional Requirements:
F.1
F.2

Input:
A commit created by an thirdparty SVN client.

Expected output:
The user receives a message from Visual Code Review indicating that it is due for code review.

Instructions:
1. Commit to the company “Visual Code Review” protected SVN server, to a branch that has a policy, following normal procedures for this action.