[TopCoder]

Code Review Tool 1.0 Component Specification

1. Design

The Tool should create baseline documentation and prepare files from CVS for code review. A baseline includes a change log and tag information.

Preparing for the code review includes creating a package (zip file) including all source code to review and some pages (html files) to sign when reviewing is done.

The tool should be configurable with parameters that define repository location, output document information (likes file names or place on disk), files that should be removed from sensitive code review package and type of review performed: full or partial.

1.1 Design Patterns

1.1.1 Strategy

Component defines several interfaces (e.g. CvsManager). Despite currently the component has only 1 implementation for each interface, in the future more implementations can be added and the strategy context can be added.

1.2 Industry Standards

CSS, CVS, HTML, Property file (for configuration), ZIP

1.3 Required Algorithms

1.3.1 Prepare review input

1) Prepare CvsManager:

1.1) Create CvsManagerImpl instance.

1.2) Populate CvsManager's directory with value of "checkoutDirectory" variable and cvsRoot with value of "cvsRoot" variable.

2) Checkout the specified projects. Project names are in "projects" variable, branches from "branches" variable should also be taken into account. Use CvsManager.checkout(project) for projects with no branch specified and CvsManager.checkout(project, branch) for projects with specified branch.

3) Generate CVS log. Use CvsManager.log() method if "logBeginDate" variable is null or CvsManager.log(beginDate) method if "logBeginDate" variable is not null. Save generated log to "log.txt" file under checkout directory.

4) Tag CVS repository. Use CvsManager.tag() method passing it with value of "tag" variable.

5) Create working folder. Its name is specified in "workingDirectory" variable. If such folder already exists, clean it up (delete all content).

6) Create baseline change log file called "ChangeLog<YYYYMMDD>.txt" (where <YYYYMMDD> is the current date in the corresponding format) located in the working folder. Open this file for writing (create OutputStream for it).

7) Create ChangeLogWriter (use default implementation – ChangeLogWriterImpl).

8) Read CVS log (generated before) and gather info from it. Use CvsLogReader (use default implementation – CvsLogReaderImpl) to read CVS log entry by entry. Each entry describes 1 file, all the checked out files will be there is the log. So, use it to construct list of all the files. For each file:

8.1) Write record to the baseline change log – call ChangeLogWriter.write() providing it with output stream for baseline change log and the current CvsLogEntry.

8.2) Determine its project name. The project name is the top-level directory in the working file path relative to the CVS log entry (CvsLogEntry.workingFile).

8.3) Determine its package name. The package is the directory right above the "src" directory (in CvsLogEntry.workingFile). If file has no package (no "src" in the path), then we are done with this file.

8.4) If filename passes "allowedFileExtensions" and "skippedNames" filters, then:

8.4.1) Create folder "CodeReview<YYYYMMDD>_<project>" (where <YYYYMMDD> is the current date in the corresponding format and <project> is the project name) under working directory (if not created yet). Let's call this folder a "project folder". Once folder is created, copy file referred by "projectCssFilename" variable to that folder.

8.4.2) Create a file "ChangeLog<package>.txt" (where <package> is the package name) inside the folder mentioned above (if not created yet).

8.4.3) Append record to the file mentioned above. Use ChangeLogWriter.write() providing it with output stream for mentioned file and the current CvsLogEntry.

9) In case of full review (variable "fullReview" is true):

9.1) In the checkout directory (recursively for subfolders), delete all files which doesn't conform the "allowedFileExtensions" and "skippedNames" filters.

9.2) In the checkout directory (recursively for subfolders), delete empty folders.

9.3) For each project, for each package:

9.3.1) Archive all remaining checked out files into "<package>.zip" (where <package> is the package name) archive, keeping the folder structure.

9.3.2) Copy file referred by "reportTemplateFilename" variable to the project folder and name it there as "CodeReview_<package>.html" (where <package> is the package name).

9.3.3) Update the HTML file mentioned above with correct data and links to "<package>.zip" and "ChangeLog<package>.txt" files.

10) In case of partial review (variable "fullReview" is false):

10.1) For each project, for each package:

10.1.1) For each file:

10.1.1.1) Determine the first and last revisions of this file as last and first (respectively) elements of CvsLogEntry.revisions list. File requires a full review if it has less than 2 revisions in that list. If more than 2, then partial review is need for that file.

10.1.1.2) If this file requires partial review, then call CvsManager.diff() with first revision as "beginRevision" and last revision as "endRevision" and current file name as filename. Read data from returned stream and append it to "<package>_difflog.txt" (where <package> is the package name) file under project folder (create diff log file if not exists yet).

10.1.1.3) If this file requires full review, then add this file to "<package>.zip" (where <package> is the package name) archive under project folder (create this archive if not exists yet).

10.1.2) If some file required full review, then move "<package>_difflog.txt" file (if exists) to "<package>.zip" archive.

10.1.3) Copy file referred by "reportTemplateFilename" variable to the project folder and name it there as "CodeReview_<package>.html" (where <package> is the package name).

10.1.4) Update the HTML file mentioned above with correct data and links to "<package>.zip" (or "<package>_difflog.txt" if no files required full review) and "ChangeLog<package>.txt" files.

11) For each project:

11.1) If there are several packages in this project, then copy file referred by "reportListTemplateFilename" variable to the project folder and name it there as "CodeReview.html".

11.2) Update the HTML file mentioned above with correct data and links to the "CodeReview_<package>.html" files for this project.

11.3) Archive the project folder into a zip file with same name as the folder plus "zip" extension. Project folder itself should not be placed into archive, only its content; folder structure should be preserved.

12) Copy file referred by "baselineLogReportTemplateFilename" variable to the working folder and name it there as "BaselineLog.html".

13) Update base line log HTML file with correct data and all links to the projects' zip files and base line change log file (use proper project names and package names; there are should be 1 row per 1 project in the table (all "Initial baseline" should point to the baseline change log and each "Code review" link should point to corresponding project zip)).

14) Archive all the content of working folder to "ResultFile.zip" in folder pointed by "resultDirectory" variable.

15) If "deleteTempFiles" variable is true, then delete all from checkout and working folders.

Notes:

1) Case insensitive comparing should be used for checking file extensions ("allowedFileExtensions" filter) and skipped names ("skippedNames" filter).
2) Algorithm should try to complete even with errors and write any encountered exceptions and inconsistency warnings to log and console. Algorithm should break (and throw corresponding exception) only if there is a fatal issue, preventing it from further processing (e.g. failed to checkout projects, failed to generate log, etc).

1.3.2 Adding files to archives

java.util.zip should be used for managing zip archives.

A typical way of adding a file to zip archive looks like this:

```
ZipOutputStream zip = new ZipOutputStream(
      new FileOutputStream("archive.zip"));
// here "path/filename.ext" is a name and path of a file inside archive,
// has nothing to do with path and name of original file
ZipEntry zipEntry = new ZipEntry("path/filename.ext");
zip.putNextEntry(zipEntry);
// here "path/filename.ext" points to original file
InputStream is = new FileInputStream("path/filename.ext");
// below is the algorithm of copying data from one stream to another,
// developer is expected to improve it (use buffer)
byte[] data = new byte[is.available()];
is.read(data);
zip.write(data);
zip.closeEntry();
zip.finish();
```

1.3.3 Templates

Provided ReportTemplate.html and ReportListTemplate.html files are templates for "CodeReview<package>.html" and "CodeReview.html" files, respectively. They contain placeholders in format %placeholderName%. Each placeholder should be substituted with value from "reportInformation" map under key equal to the placeholder name.

There are some special placeholders which are not taken from configuration, but specified by the component:

reviewDate – the current date in YYYY-MM-DD format;

reviewDateCompact – the current date in YYYYMMDD format;

packageName – the package name;

projectName – the project name;

reviewType – "Full" if "fullReview" variable is true, "Changes" otherwise;

tagName – the tag name from "tag" variable.

If there is no value available for some placeholder, use "N/A" (without quotes) value for it.

1.3.4 Logging

Logging should be performed by CodeReviewManager.prepareReviewInput() method using Log4j. All errors (ERROR level) and actions (INFO level) should be logged. Developers are encouraged to log more low level debug information at DEBUG level.

1.4 Component Class Overview

1.4.1 com.ibm.ps.codereview

CodeReviewManager

This is the main class of the component. It declared method prepareReviewInput() which performs the main component operation. So, it can be used from Java code. Also, it define a command line entry (static main() method), so the component can be used from command line.

java.util.zip means are utilized for archiving and Log4j is utilized for logging.

This class is immutable and thread-safe (thread-safe usage of affected files and CVS repository is the invoker's responsibility).

CvsManager

This is a contract for CVS manager. It defines methods for CVS operations, specifically the "checkout", "log", "tag" and "diff" operations. It also defines setters for configuring the working directory (the directory to which the projects will be checked out, where the files for log generation will be looked for, etc) and CVSROOT.

Implementations of this interface are not required to be thread-safe.

CvsLogReader

This is a contract for CVS log reader. It defines method for reading next entry (described by CvsLogRevisionEntry class) from the given log stream.

Implementations of this interface are not required to be thread-safe.

ChangeLogWriter

This is a contract for change log writer. It defines method for writing info to given CVS log entry (described by CvsLogEntry class) to the given change log stream (format is up to implementations).

Implementations of this interface are not required to be thread-safe.

CvsLogEntry

This class describes a CVS log entry.

It has empty default constructor and several fields, each with getter and setter. Setters don't perform any validation.

This class is mutable and not thread-safe.

CvsLogRevisionEntry

This class describes a revision entry of CVS log entry.

It has empty default constructor and several fields, each with getter and setter. Setters don't perform any validation.

This class is mutable and not thread-safe.

1.4.2 *com.ibm.ps.codereview.cvs*

CvsManagerImpl

This implementation of CVS manager interface is based on delegating CVS operations to CVS command line tool (path to which should be configured to cvsCommand property). It defines methods for CVS operations, specifically the "checkout", "log", "tag" and "diff" operations. It also defines setters for configuring the working directory (the directory to which the projects will be checked out, where the files for log generation will be looked for, etc) and CVSROOT.

This class is mutable and not thread-safe. In order to use it thread safely, invoker should not call setters at the same time with business methods (thread-safe usage of checked out files and CVS repository is the invoker's responsibility).

1.4.3 *com.ibm.ps.codereview.log*

CvsLogReaderImpl

This is a default implementation of CVS log reader. It defines method for reading next entry (described by CvsLogRevisionEntry class) from the given log stream.

The format of log entry follows:

RCS file: {rcsFile}

Working file: {workingFile}

head: {headRevision}

branch: {branch}

locks: {ignore}

access list: {ignore}

keyword substitution: {ignore}

total revisions: {ignore}; selected revisions: {ignore}

description: {description}

----

revision {revision.revision}

date: {revision.date}; author: {revision.author}; state: {ignore}; lines: {ignore};

{revision.comment}

----

revision {revision.revision}

date: {revision.date}; author: {revision.author}; state: {ignore}; lines: {ignore};

{revision.comment}

==========
=

So, an entry is identified as line between "RCF file..." and "===...". Revision entries are located between "---..." and either "---..." or "===..." lines. There are could be 0 or more revision entries, each should be populated to CvsLogEntry.revisions list in the same order as they go in the log.

The text between curly brackets refers to the field name (of CvsLogEntry or CvsLogRevisionEntry (if prefixed with "revision.")) to which the corresponding value should be populated.

The {revision.date} is formatted as per current locale.

This class is immutable and thread-safe as long as invoker uses input parameters thread safely.

ChangeLogWriterImpl

This is a default implementation of change log writer. It defines method for writing next entry (described by CvsLogEntry class) to the given log stream.

The format of log entry looks like this:

{project1}<tab>{path}<tab>{revision}<tab>{date}<tab>{author}<tab>{comment}

{project1}<tab>----------<tab>{revision}<tab>{date}<tab>{author}<tab>{comment}

{project1}<tab>----------<tab>{revision}<tab>{date}<tab>{author}<tab>{comment}

{project2}<tab>{path}<tab>{revision}<tab>{date}<tab>{author}<tab>{comment}

{project2}<tab>----------<tab>{revision}<tab>{date}<tab>{author}<tab>{comment}

{project2}<tab>----------<tab>{revision}<tab>{date}<tab>{author}<tab>{comment}

If project has zero revisions, it will not be printed to the log. If project has more than 1 revision, for 2nd and next revisions "----------" will be printed instead of path.

{projectX} states for the name of the project which is determined as a top-level directory in the working file path relative to the CVS log entry (CvsLogEntry.workingFile).

{path} states for the working file path relative to the project directory (the CvsLogEntry.workingFile without top-level directory).

{revision} states for file revision (CvsLogRevisionEntry.revision).

{date} states for date of file revision (CvsLogRevisionEntry.date) in "YYYY/MM/DD hh:mm:ss" format.

{author} states for file revision author (CvsLogRevisionEntry.author).

{comment} states for file revision comment (CvsLogRevisionEntry.comment).

<tab> states for a TAB character.

This class is immutable and thread-safe as long as invoker uses input parameters thread safely.

1.5 Component Exception Definitions

1.5.1 com.ibm.ps.codereview

CodeReviewToolException

This is a base exception of the component. It indicates general issue with code review tool operating.

This class inherits from not thread-safe class, so it's not thread-safe.

CvsOperationException

This exception indicates error with executing CVS operation.

This class inherits from not thread-safe class, so it's not thread-safe.

CvsLogException

This exception indicates error with CVS log (reading, parsing, etc).

This class inherits from not thread-safe class, so it's not thread-safe.

ChangeLogException

This exception indicates error with change log (writing).

This class inherits from not thread-safe class, so it's not thread-safe.

CodeReviewToolConfigurationException

This runtime exception indicates fatal issue with component configuration.

This class inherits from not thread-safe class, so it's not thread-safe.

1.6 Thread Safety

The component, being used via CodeReviewManager, is thread-safe (though thread-safe usage of affected files and CVS repository is the invoker's responsibility).

Though the interfaces defined in this component don't require their implementations to be thread-safe. Also, the component contains not thread-safe data-transfer classes and exceptions.

2. Environment Requirements

2.1 Environment

2.1.1 Development environment

Development language: Java 1.6

Compile target: Java 1.6

2.1.2 QA environment

Solaris 7

RedHat Linux 7.1

Windows 2000

Windows 2003

Ubuntu 8.04

Ubuntu 10.04

2.2 TopCoder Software Components

Not allowed.

2.3 Third Party Components

Log4j 1.2.16: http://logging.apache.org/log4j/1.2/download.html

CVS 1.11.22: http://download.savannah.gnu.org/releases/cvs/binary/stable/

NOTE: The default location for 3rd party packages is ../lib relative to this component installation. Setting the ext_libdir property in topcoder_global.properties will overwrite this default location.

3. Installation and Configuration

3.1 Package Name

com.ibm.ps.codereview

com.ibm.ps.codereview.cvs

com.ibm.ps.codereview.log

3.2 Configuration Parameters

CodeReviewManager is configured via properties file.

Parameter	Description	Value
checkoutDirectory	The directory to which the files from CVS repository will be checked out.	String. **Required.** Not empty.
cvsRoot	The CVSROOT of CVS repository from which the data should be taken.	String. **Required.** Not empty.
projectsCount	The amount of projects to be checked out.	int. **Required.** Positive integer.
projectsX	Name of X-th project which should be checked out and for which the code review	String. **Required.**

	package should be prepared. X is an integer between 1 and projectsCount, inclusive.	Not empty. projectsX parameter is required for each X in the mentioned range.
branchesX	This name of X-th branch assigned to projects in the "projects" list. X-th element of this array is the branch for the project corresponding to the X-th element in the projects array.	String. Optional. Not empty. If X-th element of this list is not specified, it means that no specific branch should be used for X-th project.
tag	The name of tag with which the checked out projects should be tagged.	String. **Required.** Not empty.
logBeginDate	The date since which the changes should be logged (and processed for review). If null, the full CVS log will be generated.	Date in YYYY-MM-DD format. Optional. If not specified, full log will be generated.
allowedFileExtensions	The list of file extensions. Only the files with extensions listed here will be passed to review. If no extensions specified, any extension is allowed. This should be a comma separated list (each extension without leading dot).	String. Optional. If not specified (or empty), any extension will be allowed. Elements (separated by commas) should not be empty.
fullReview	Indicates which type of review should be performed: true for full review and false for partial review.	boolean. Optional. Default value is true.
deleteTempFiles	Indicates if to delete the temporary files.	boolean. Optional. Default value is false.
skippedNames	The list of names. The files which contain ant of these names into their path (into either a file name or a directory name, the path is relative to the package). This should be a comma separated list.	String. Optional. Default value is no skipped names. Elements (separated by commas) should not be empty.
resultDirectory	The directory to which the code review package should be put.	String. **Required.** Not empty.
projectCssFilename	The project CSS filename.	String. **Required.** Not empty.
reportTemplateFilename	The report template filename.	String. **Required.** Not empty.
reportListTemplateFilename	The report list template filename.	String. **Required.** Not empty.
baselineLogReportTemplateFilename	The baseline log report template filename.	String. **Required.** Not empty.
reportInformationCount	The amount of reportInformation key-value pairs.	int. Optional.

		Not negative. Default is 0.
reportInformationKeyX	The X-th key of reportInformation map. A key is a marker in template file. X is an integer between 1 and reportInformationCount, inclusive.	String. **Required (for X in [1, reportInformationCount] range).** Not empty.
reportInformationValueX	The X-th value of reportInformation map. A value is a value with which the corresponding marker will be substituted. X is an integer between 1 and reportInformationCount, inclusive.	String. **Required (for X in [1, reportInformationCount] range).**
workingDirectory	The directory to which the code review package files will be put before archiving. Temporary files will also be put there.	String. **Required.** Not empty.
loggerName	The name of the Log4j logger to be obtained by invoking Logger.getLogger(String).	String. Optional. Not empty. Default value is a full class name.
cvsCommand	CVS command for executing CVS.	String. Optional. Not empty. By default, default CVS command in CvsManagerImpl will be used.

3.3 Dependencies Configuration

None.

4. Usage Notes

4.1 Required steps to test the component

- Extract the component distribution.
- Follow Dependencies Configuration.
- Execute 'ant test' within the directory that the distribution was extracted to.

4.2 Required steps to use the component

Follow demo.

4.3 Demo

4.3.1 Sample configuration file

```
checkoutDirectory=/usr/home/checkout
cvsRoot=:pserver:myserver@myhost.mydomain.com:/rep
projectsCount=1
projects1=APP
tag=TAG_20101228
allowedFileExtensions=java,js,xhtml
fullReview=false
deleteTempFiles=true
resultDirectory=/usr/home/codereview
projectCssFilename=/usr/home/templates/project.css
reportTemplateFilename=/usr/home/templates/ReportTemplate.html
reportListTemplateFilename=/usr/home/templates/ReportListTemplate.html
```

```
baselineLogReportTemplateFilename=/usr/home/templates/BaselineLogReportTem
plate.html
reportInformationCount=3
reportInformationKey1=author
reportInformationValue1=Jhon Do
reportInformationKey2=versionNumber
reportInformationValue2=6.22
reportInformationKey3=managerFullName
reportInformationValue3=Jhon Order
workingDirectory=/usr/home/temp
```

4.3.2 *Command line usage*

- `java CodeReviewManager -h`

Help info is displayed.

- `java CodeReviewManager /usr/home/cfg/configuration.prop`

Assuming that "/usr/home/cfg/configuration.prop" points to file listed in 4.3.1, the code review input will be produced according to configuration.

4.3.3 *API usage*

```
CodeReviewManager manager =
      new CodeReviewManager("/usr/home/cfg/configuration.prop");
manager.prepareReviewInput(false);
```

Result is the same as after invoking last command from 4.3.2.

4.3.4 *Sample output*

The content of the "usr/home/codereview" folder will look like this:

-- ResultFile.zip

-- -- BaselineLog.html

-- -- ChangeLog20101228.txt

-- -- CodeReview20101228_APP.zip

-- -- -- projectstyle.css

-- -- -- CodeReview.html

-- -- -- CodeReview_util.html

-- -- -- ChangeLogutil.txt

-- -- -- util_difflog.txt

5. Future Enhancements

Review input preparation algorithm can be changed (file formats, code review package content and structure, etc).