Code Review Preparation Tool Requirements Specification

1. Scope

1.1 Overview

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.

1.1.1 Version

1.0

1.2 Logic Requirements

The tool should implement the following workflow:

- Create change log – see section 1.2.1
- Tag repository – see section 1.2.2
- Prepare for code review - see section 1.2.3

The last step can be done for a full review and a partial review (only changes from last review done). Prior starting these steps the tool must be able to access needed repositories on file system. This can be done by checking out all files in one directory and use it as base for all other operations.

1. Command example:
   `cvs co APP/Application/TTT/project`
2. This operation should support branches also:
   `cvs co -r CODE_BRANCH  APP/Application/TTT/project`

Tool requires all files checkout from repository to local file system when preparing sensitive code review [1.2.3]. Files that have been changed should be packaged. Tool can checkout files in several directories, but it's easier to have them all in one directory on file system and use later for preparing packages. Tool should invoke command line tools and checkout all files. Command lines tools should also be used for tagging repository.

1.2.1 Create change log

This step creates a log file from CVS repository. The log can be produced based on 3 conditions:

- Full change log: *cvs log -N > log.txt*
- Partial change log: *cvs log -N -d"YYYY-MM-DD<" > log.txt"*
- Partial change log for branch: *cvs log -N -r BRANCH -d "YYYY-MM-DD<" > log.txt*

The change log type, date and branch name must be defined as tool's parameters.

Produced log by CVS will be in following format:

```
RCS file:
/repository/APP/Application/TTT/ttt/src/com/tdd/web/util/
AppUtil.java,v

Working file:
APP/Application/TTT/ttt/src/com/tdd/web/util/AppUtil.java

head: 1.6

branch:

locks: strict

access list:

keyword substitution: kv

total revisions: 10; selected revisions: 2

description:
```

```
----------------------------

revision 1.1.2.4

date: 2010/09/10 11:06:48;  author: NAME;  state: Exp;
lines: +3 -6

changed status after fail in app
----------------------------

revision 1.1.2.3

date: 2010/09/10 09:07:52;  author: NAME;  state: Exp;
lines: +2 -2

changed redirect page
=============================================================
======
```

This file must be transformed into the following format:

```
APP
APP/Application/TTT/ttt/src/com/tdd/web/util/AppUtil.java
1.1.2.4   2010/09/10 11:06:48      NAME  changed status
after fail in app
```

Use the “TAB” character between the revision number and the date as seen in above example line

Revision 1.1.2.3 should also be included in the formatted log file. Please see below for an example of a log file with two files.

The formatted file must filter only entries, which have revision changes. The files with no revisions shouldn't be added. After this step the tool must output the original log file and also the formatted log file.

Below is an example of log file with two files: one file with 3 revisions and one file with 0 revisions. Files that have 0 revisions should not be added to formatted log file.

```
RCS file:
/repository/APP/Application/TTT/ttt/WebContent/leastcostselect.
xhtml,v
Working file:
Application/TTT/ttt/WebContent/leastcostselect.xhtml
head: 1.6
branch:
locks: strict
access list:
keyword substitution: kv
total revisions: 6;    selected revisions: 3
description:
----------------------------
revision 1.6
date: 2010/11/03 10:51:51;  author: LT112233;  state: Exp;
lines: +1 -3
removed unnecessary condition
----------------------------
revision 1.5
date: 2010/10/29 09:53:40;  author: LT332211;  state: Exp;
lines: +6 -4
changed ATN logos display
----------------------------
revision 1.4
date: 2010/10/15 11:24:54;  author: LT112233;  state: Exp;
lines: +17 -2
LT109370 added logos for credit card brand and
AvailableNetworks, changed the main text, button labels, button
style
=============================================================
==============

RCS file:
/repository/APP/Application/TTT/ttt/WebContent/paymentach.xhtml
,v
Working file: Application/TTT/ttt/WebContent/paymentach.xhtml
head: 1.10
branch:
```

```
locks: strict
access list:
keyword substitution: kv
total revisions: 11;    selected revisions: 0
description:
============================================================
==============
```

Formatted log file:

```
PS6  Application/TTT/ttt/WebContent/leastcostselect.xhtml
1.6        2010/11/03 10:51:51       LT112233  removed unnecessary
condition
PS6  -------------
1.5        2010/10/29 09:53:40       LT332211  changed ATN logos
display
PS6  -------------
1.4        2010/10/15 11:24:54       LT112233  LT109370 added logos
for credit card brand and AvailableNetworks, changed the main
text, button labels, button style
```

Please see attachment Updated[20101117].txt with the following example.

1.2.2 Tag repository

The repository needs to be tagged after first step is completed. The tag must be configurable in tool as a parameter.
Tagging can be done with CVS command: `cvs tag CODE_20100310`

1.2.3 Prepare for code review

This step analyses checkout files and produces output that can be used for review. The tool should implement these steps:

1. Create a working directory
2. The template use a style sheet for formatting so copy this one: `projectstyle.css` to working directory. (See appendix for attached file).

3. The changed files are divided into different documents corresponding to the components (packages). For each package there should be a list of changes included in development period covered by the review. The package name is always the name of repositories directory which is above "src" directory. (e.g. APP/tool/src/pak must be associated as "tool" package).
   The lists of changes should be made from the baseline change log.
   a. Copy all changes from the baseline change log for first package into *ChangeLog<package name>.txt.*
   b. The new change log file must contain only file entries that are defined in tools parameters (e.g. *.java, .xml* etc.).
   c. Repeat step *a* and *b* for all other packages.
5. If this is a ***partial*** (changes only from last baseline) review then tool must aggregate all partial changes for the package in a single text file. The following steps must be taken:
   a. create a command list by adding a comparison statement for each file needed to compare
      "*cvs diff –u –r 1.a –r 1.b <filename> <stream characters> <package name>_difflog.txt*
      where *<stream characters>* is:
      - *o >* for the first file in the package
      - *o >>* for following files in the package

      <filename> should be fully qualified by path (copy full name including path from baseline change log). Revision numbers can also be found in the change log. The first revision number is the revision reviewed at the last review and the second revision number is the latest checked in revision;
   b. execute the command list in console;
   c. copy all **_difflog.txt* files to the working directory with the other code review files;
   d. copy *CodeReview_Template.html* (see appendix for attached file) into *CodeReview_<package name>.html;*
   e. update *CodeReview_<package name>.html* with correct package name and links to *<package name>_difflog.txt* and *ChangeLog<package name>.txt* files.

   If some files needs *full review* (never reviewed files before) and others only need difference review then make the *<package name>_difflog.txt* as described above and add it to a *<package name>.zip* file with the files that need full review. In this case the link should be to the *<package name>.zip* file.

Values for the first and second revision number can be found in change log or in formatted changed log. Example:

```
PS6  Application/TTT/ttt/WebContent/leastcostselect.xhtml
1.6        2010/11/03 10:51:51       LT112233  removed unnecessary
condition
PS6  -------------
1.5        2010/10/29 09:53:40       LT332211  changed ATN logos
display
PS6  -------------
1.4        2010/10/15 11:24:54       LT112233  LT109370 added logos
for credit card brand and AvailableNetworks, changed the main
text, button labels, button style
```

In that case, the command should be written like this:

```
cvs diff -u -r 1.4 -r 1.6
Application/TTT/ttt/WebContent/leastcostselect.xhtml >>
TTT_difflog.txt
```

6. If this is a ***full*** review all relevant files in the root package directory must be included for review. Use the following steps:
   a. clean-up the directory with all checked out files;
      i. all files that are not type of specified type in parameters (e.g. if *.classpath* is not specified as parameter) must be removed;
      ii. all empty folders should be removed;
   a. for each package copy all remaining files into a *<package name>.zip* file. Keep the folder structure. Package is always the directory above "*src*" directory;
   b. copy *CodeReview_Template.html* (see appendix for attached file) into *CodeReview_<package name>.html*;
   c. update *CodeReview_<package name>.html* with correct package name and links to *<package name>.zip* and *ChangeLog<package name>.txt* files. Information which is updated (besides links), must be customizable. Customizable parameters will be listed in section 3.1.
7. If there are several packages then copy the template *CodeReviewTemplate.html* to the working folder for the current review. Save it as *CodeReview.html*. Update *CodeReview.html* to include all links to the *CodeReview_<package name>.html* files.
8. Collect all files supporting the review into a *CodeReviewYYYYMMDD_xxx.zip* file where *YYYYMMDD* is the date the review is prepared and *_xxx* is the sub area extension

(must be customizable with parameters). This zip file is the final output that tool should produce. (see output example in appendix).

9. Work files like not formatted changelog should not be deleted, unless specified in parameter.
10. During last step tool should produce BaselineLog report (see appendix for template and example). This report should contain links to produced result zip file and baseline CVS log text file with current dates (date format *YYYYMMDD*).

1.3 Required Algorithms

Algorithm may be created or chosen by developer. This tool should not break if there are any inconsistencies in repository or configuration. It should try to complete even with errors and write any encountered exceptions and inconsistency warnings into log file and console.

1.4 Example of the Software Usage

Tool use should be straightforward and follow scenario below:

- Define parameters
- Start work flow
- Monitor progress information of flow
- See output result in opened directory after the flow is finished

1.5 Future Component Direction

None.

2. Interface Requirements

2.1.1 Graphical User Interface Requirements

The GUI can be command line or graphical interface. The main requirement is that parameters can be edited, changed, saved and reloaded easily (like opening a document and changing it). Also tool must show status on its progress or problems if it encounters any (can be done in console or GUI).

2.1.2 External Interfaces

None.

2.1.3 Environment Requirements

- Development language: Java1.6
- Compile target: Java1.6

2.1.4 Package Structure

com.ibm.ps.codereview

3. Software Requirements

3.1 Administration Requirements

3.1.1 What elements of the application need to be configurable?

The Tool must specify the following parameters:

1. *directory* – check-outed source code location
2. *host* – host name and rood directory of repository. CVSROOT value (*:pserver:NAME@host.ibm.com:/APP/rep*)
3. *repository* list – list of repositories which should be checked out for code review preparation. For example:

NAME/Application/ProjectType/package1,
NAME/Application/ProjectType/package2

The repository should also support branch checking out, that would need additional parameter with branch name directly associated with a single repository (location_branch).

4. *tag* – tag code that should be used for tagging checkout files (*TAG_201015_T*).
5. *change log* – this should define if its a full change log (no date specified), partial changes before date (<2009-10-11), partial change log for branch (include also branch name when creating the log).
6. *allowed file types* – list of file types that are allowed (*java, js, xhtml*).
7. *review type* – full or partial (this changes the workflow).
8. *delete temp files* – flag for setting if temp files like checkouted and primary CVS log should be deleted.
9. s*kipped names* – list of names, that tool should not include in review changes if a file or a directory contains one of these names (*test, dummy*).

10. *result directory* – location where all output is produced
11. *report template* – location of report template which is used for producing report files for each package (*/home/test/CodeReview_template.html*).
12. *report list template* – location of report which includes link list too all package reports. (*/home/test/CodeReview.html*).
13. *baseline log report template* – overall report with information and links to review result file and baseline log file. (*/home/test/BaselineLogTemplate.html*)
14. *report information* – should define a list of key-value pairs for customizable information used in report. The template might introduce new parameters so this list must dynamically find and replace them (*e.g. authorFullName/Name Lastname,versionNumber/6.2.2,managerFullName/Name Middlename Lastname,reviewDate/13-09-2010,reviewerFullName/Name Lastname,approveDate/2010-09-13*). Please note that report is not related to change log. Please see documents *CodeReview.html*, *CodeReview_template.html* and *BaselineLogTemplate.html*. Values that should be changed are marked in % symbols.

3.2 Technical Constraints

3.2.1 Are there particular frameworks or standards that are required?

- Java
- Eclipse 3.4 IDE (project compatible)

3.2.2 TopCoder Software Component Dependencies:

Do Not Include Any TopCoder Generic components in this design.

3.2.3 Third Party Component, Library, or Product Dependencies:

Tool should be able produce zip files, and access cvs repository.

3.2.4 QA Environment:

- Solaris 7
- RedHat Linux 7.1

- Windows 2000
- Windows 2003
- Ubuntu 8.04
- Ubuntu 10.04
- CVS

3.3 Design Constraints

The component design and development solutions must adhere to the guidelines as outlined in the TopCoder Software Component Guidelines. Modifications to these guidelines for this component should be detailed below.

3.4 Required Documentation

3.4.1 Design Documentation

- Use-Case Diagram
- Class Diagram
- Sequence Diagram
- Component Specification

3.4.2 Help / User Documentation

There should be a short tutorial, on how to use the tool, what parameters can be defined and their meaning. Also there should be a list of possible errors, that user may expect and know how to act on them.

4. Appendix

4.1 File examples

- BaselineLogTemplate.html
- CodeReview.html
- CodeReview_template.html
- ResultFile.zip