METCS673

PLM - Project Lifecycle Management

Software Confiugration Management Plan

Author:
Christian Heckendorf

October 7, 2013

1 Proposed Configuration Management Plan

1.1 Configuration Items and Tools

All configuration items will be stored in a git repository hosted on GitHub. The following configuration items are required:

- Documentation directory
  - Planning documents
  - Code documentation
  - User level documentation
- Front-end code directory
  - User interface code
- Back-end code directory
  - Services code
- Database code directory
  - Database create scripts
  - Sample insert scripts

Only essential files required for QA, deployment, or understanding (documentation) should be added to the repository. In general, binaries and temporary files should not be added. Binary documentation is allowed in the repository but the plain text source documents should accompany them.

1.2 Change Management and Branch Management

The general outline is as follows:

- The master branch should be at demo quality at all times.
- Unstable code can be merged into a development branch.
- New features should go in their own branches, which will at some point be merged into the development branch.
- Approximately every three weeks, the development branch will be merged back to the master branch to prepare for the presentation.

This system should allow the team to always have a state available for a demo while continuing to add features in feature branches and fixing bugs in the development branch. To begin, one person should commit the basic layout of the project to the master bracnch and create a development branch.

From there, each member can begin working on feature branches based on the new development branch. Each feature should have its own branch and each branch should be given a short yet descriptive name representing the feature. At some point, a feature will be complete and can be merged back to the development branch. QA may want to review the code before merging to minimize bugs creeping into the development branch. At the end of a development iteration, the development branch will be merged back to the master branch to prepare for the presentation.

1.3 Code Commit Guidelines

Each commit should contain as few independent changes as possible and should have a descriptive commit message. Code that produces validation or build errors should not be pushed to GitHub. Broken code may be committed to local repositories but the additional commits required to produce buildable code must also be present in the repository before it is pushed to GitHub.

2 Reference

2.1 Git Basics

Since I don't know what git client you'll be using, I'll summarize the basic command line utility commands.

When you first start working on the project, you'll want to make a local copy of the repository to keep track of your changes:

```
git clone https://github.com/heckendorfc/plm.git
```

Before you begin working, you should create a feature branch based on the development branch:

```
git checkout -b somefeature origin/development
```

At any point, you can check the status of your local repository. This will tell you what files have been changed, what branch you are using, and other things:

```
git status
```

Once you have some code written and want to save your work to the repo, tell git which files you would like to commit to the repo and commit them:

```
git add file1 file2 ...
git commit -m "commit message here"
```

To get the latest code that other team members have shared on the github repo:

```
git fetch origin
git pull
```

After you have one or more commits in your local repo, you'll want to share them with the rest of the team:

```
git push -u origin somebranch
```

Further reading: http://git-scm.com/book