Version management document

PmoC
13.4.2003
Version 1.6

T-76.115 Version management document
PmoC

Version	Date	Author	Description
1.6	11.4.2003	Lönnberg	Added DE phase experiences and overview of entire project.
1.5	24.3.2003	Lönnberg	Added locking solution to test data problem. Restructured description of solutions somewhat.
1.4	21.3.2003	Lönnberg	Noted I3 phase experiences
1.3	6.2.2003	Lönnberg	Noted I2 phase experiences
1.2	29.11.2002	Lönnberg	Revised to reflect experiences
1.1	28.10.2002	Lönnberg	Revised based on mentor feedback
1.0	25.10.2002	Lönnberg	Sent to customer for checking

Table of contents

1. Introduction

This introduction is based on a short description of version management in Haikala's and Märijärvi's book on Software Engineering[1], our own experiences and Cederqvist's documentation for CVS[4].

When developing a software product, developers often find that they need to make further changes to an older version of a product, for example if an error needs to be fixed and the customer can not use the latest version. Similarly, when developing a major product, it is often useful to be able to test and stabilise one release while working on additional features for the next release.

Also, when one has many developers working separately at odd hours, co-ordinating their activities becomes quite hard. It is therefore necessary to have some sort of system that can keep track of and merge changes to a program made by different people simultaneously.

Version management exists to give developers the ability to efficiently handle different *versions* of a product. A version is essentially a snapshot of what the project looks like at a given moment of time. Usually, versions follow each other linearly and are in this case called *revisions*. However, sometimes one wishes to create two parallel versions based on the same version; this is called *branching* or *forking*, and the resulting parallel revision chains are known are *branches* or *forks*. All versions are stored in a centralised *repository*.

2. Requirements of our project

In our project, we have 7 people working more or less independently, as each one has different other tasks such as jobs and other courses. For this reason it is important for us to be able to work separately on different parts of the product without the overhead of explicit co-ordination. For this to be feasible, a solution has to be found that allows everyone to maintain a private copy of the product, hack it about without caring what anyone else is doing, and if the result is satisfactory distribute the changes to everyone else working on the same branch. It may also be useful to fork the project to be able to develop new features while stabilising; this saves time, which is often a critical resource.

Obviously, the version management solution that we apply must also be one that we can use in practice; it must be available to us wherever we work at zero or low cost in both time and money.

3. Possible solutions

As we are primarily working using the computers provided by SoberIT in room A218[2] of the HUT CS building, our choice of tools is limited to those that are already installed, those that we can install without Administrator privileges, and those that we can convince SoberIT support personnel are necessary.

A218 computers only appear to have one version management tool: WinCVS. Earlier, Microsoft SourceSafe appeared to be installed on these machines, but I can no longer find any trace of it; it appears to have been removed when a software upgrade was made in the beginning of October. As SourceSafe licenses are somewhat expensive

(on the order of 50 euros per user), it is unlikely that we can convince SoberIT to acquire licenses unless SourceSafe provides critical functionality.

WinCVS[3] is a client for the popular CVS version management system[4], of which several free implementations exist. CVS client and server software is also installed on both HUT-CC Unix and Niksula machines, which all members of the development team can access. Further, CVS clients are available freely for most operating systems, making it easy for project members to install use CVS at home or on other systems they may use (e.g. HUT-CC Windows).

CVS provides all the features mentioned in the above requirements and is quite well documented. Further, the members of the development team have used CVS previously. Some have administered CVS repositories before.

As CVS provides all the required functionality and is both familiar and available, we see no reason to further consider other solutions.

4. Implementing CVS

Implementing CVS in our project can be boiled down to four aspects: creating the repository, setting up user access to the repository, distributing revisions, and generating releases. Jan Lönnberg is responsible for CVS maintenance and support.

4.1 Creating the repository

The CVS repository can theoretically be placed anywhere on the Internet accessible to us; in practice, it is easiest to have it somewhere where we have the ability to choose who can access it. As HUT-CC provides a convenient way for normal users to define groups that have access to certain files, it is easiest to place the repository on the HUT-CC Unix home directory filesystem. This can be done using standard repository creation procedures on any HUT-CC Unix machine.

4.2 Setting up user access to the repository

Most CVS clients (including WinCVS and command line CVS) can be used more or less immediately with the CVS repository created above using the following CVS root:

```
username@kosh.hut.fi:/u/opi/01/jlonnber/CVS/
```

Here, `username` is the name of the user on HUT-CC, and `kosh` must be used if the system is accessed from outside the HUT network.

However, as rsh logins are not allowed by HUT-CC, setting the CVS root will not suffice. We also need to specify that authentication is by SSH server and that we want to use an alternative to rsh; the ssh2 client; in A218 this is `c:\program files\f-secure\ssh\ssh2.exe`, on most Unix systems it is `ssh`. The exact mechanism by which the settings above are made depends on the exact program used. In WinCVS, all of the settings mentioned here are found in the preferences dialogue. In command-line CVS, the option `-d` is used to specify the CVS root (only needed when checking out) and SSH is used to replace RSH by setting the environment variable `CVS_RSH` to `ssh` (or whatever the SSH client is called).

4.3 Distributing revisions

From here on, the developers can check in and out files as they please using WinCVS (or whatever CVS tool each developer prefers). However, at this stage a little discipline is required to prevent one person from wrecking another's work. Before committing a change to the repository, a developer should at least perform a cursory check that the change does not break any major functionality. Specifically, the committed code should at least compile and run sufficiently well to avoid the situation where the current revision is broken and everyone has to wait until it is fixed (or, even worse, all fix it in different ways at the same time).

4.4 Generating releases

Release versions of the product can easily be generated by exporting the source tree from the CVS repository and building it. Branches and tags can be used to identify the version to be released if development on other features is also in progress.

5. Files to be stored in CVS

Obviously, the source code should be stored in CVS. Similarly, storing object code in CVS is wasteful and usually useless. Documentation is a more interesting case; anything written in plain text or some form of plain text with markup (e.g. HTML or LaTeX) can be stored in CVS with full merging and version control functionality. HTML, was, however ruled out due to its lack of proper layout control and LaTeX was ruled out due to lack of experience among some members of the group. Instead, we have used Microsoft Word to produce documents. Word files can only be stored in CVS meaningfully as binary files, which means that most of the advantages of CVS are lost. For example, merging of changes is not possible using Word files in CVS. To avoid conflicts one may need to use reserved checkouts. This problem can be circumvented by writing initial versions of documents in plain text or markup formats (e.g. HTML, LaTeX or RTF), and switching to Word format only for final formatting and conversion to PDF. Alternatively, the documentation can be stored outside CVS, e.g. in the shared directories provided on the server in A218 (this is easily done under NT, and Word provides limited locking facilities to prevent different people from editing the same document at the same time), or stored separately for each person and shared by email (which is incredibly messy and should be avoided).

6. Experiences

6.1 At end of PP phase

At this stage of the project (near the end of project planning), the CVS repository has been created and it has been accessed from some of the machines in A218. No meaningful documentation has yet been submitted to the repository, as most people felt that writing documents in plain text, LaTeX or HTML and then converting to Word is unnecessarily complicated, and CVS was not perceived as providing any benefits when Word was used. Similarly, the repository does not contain any code at this stage. Distribution and synchronisation of documents was done using shared directories on the A218 directory server and email, and in practice this resulted in a few conflicts that had to be merged manually that could have been handled automatically if CVS and a suitable document format had been used. Some of the

emails were sent to the wrong email addresses, leading to asynchrony in document versions between the developers and a lot of confusion.

Creating the CVS repository and setting up access with WinCVS took roughly an hour; the settings required in WinCVS to access the repository using SSH were not entirely obvious.

6.2 After end of implementation 1 phase

In general, the response to CVS during the first implementation phase has been favourable. Using CVS at home and at SoberIT presented little problems. In particular, the developers were pleased that they could start using the CVS from the beginning of implementation. All developers have throughout the implementation process checked out code, updated and committed code with very little trouble. The CVS repository now contains dozens of files and is in day-to-day use.

A few minor problems did crop up. Once, we had to recreate the repository from scratch to make package names conform to our coding standards. As CVS does not support renaming of files or directories, the entire directory structure had to be copied. This caused minimal disruption, as the change was made at quite an early stage and at a time of day with little development activity. This sort of problem could have been avoided by reading the coding conventions document before creating lots of directories.

When using PCs at the Maarintalo, some of us initially had problems getting CVS to connect. This problem was caused by people not following the instructions in this document properly (usually because they had not read the document).

We noticed that some developers committed code to the CVS and updated their local copies very seldom, which made it harder to integrate their code with that written by others. Ideally, developers should synchronise as often as possible.

We have decided to avoid branching the repository to avoid problems when merging the branches.

These experiences suggest that the CVS itself works fine and needs no modification. A little more effort should have been spent on educating the developers about the system.

The CVS system was only used for source code and plain text documentation files. The developers are still reluctant to use the CVS for Word files; instead, the Word files are distributed using both SoberIT's NT file server and the group's web forum[5]. This solution is not entirely satisfactory, as it is not entirely clear which version should be used: the version on the forum or the version on the file server? This problem is compounded by disagreements on which system should be used as the primary repository for documents. We plan to move to using the forum as our primary document repository to avoid this problem.

6.3 After end of implementation 2 phase

When working in Maarintalo, we discovered that checking out from CVS to local disk is not a good idea, for the simple reason that local disks on the Maarintalo

workstations are not accessible remotely. Thus, if someone else is using “your” workstation, you can’t access your own copy of the code (with your modifications, settings and such). Trying to use someone else’s checked out copy results in some rather nasty file permission problems that can easily lead to corrupting the local copy (which is especially dangerous if the local copy has uncommitted changes). Thus, new checkouts from CVS to Maarintalo workstations should be done to a private network shared directory; preferably one’s home directory. However, this still does not solve the problem of maintaining compilation settings for AnyJ; even with CVS checkouts on a network share, the time-consuming activity of setting up AnyJ for one’s own copy of the source code remains. Theoretically, one could store AnyJ settings in the CVS; in practice, this causes a lot of problems due to e.g. absolute file paths, which would limit this system to one copy of the source per workstation, which is at least as bad as the current system.

In Windows XP, it ought to be possible for several users to be logged in at once. However, HUT-CC appears to have disabled this feature for some unfathomable reason on their Windows workstations. On the Unix workstations, access to the local disks is limited to writing temporary files (deleted on logout, apparently), and on both the Windows and Unix systems, the network share quota is too small to store installations of AnyJ and the required libraries. The problem here is the use of AnyJ; makefiles and compilation scripts are commonly stored in CVS with practically no problems. With an alternative build environment (e.g. Windows batch files instead of AnyJ settings), we could store all the libraries in CVS and allow quick checking out of everything necessary to any machine with a Java SDK. However, changing build environment from the AnyJ IDE to an alternative at this stage will cause more trouble than it’s worth. Alternatively, if everyone had a personal workstation, everything would be just fine.

Synchronisation to the CVS appears to be more frequent now, making it easier to keep track of activity and coordinate work.

Use of SoberIT’s facilities has declined to the point where the documents stored on their NT file server are inaccessible most of the time. Because of this, the group’s discussion forum is now the only distribution point for documents, which means that the latest revision of a document is always available at the forum. This development removes the need to check two places to find the latest version of a document. The forum is more accessible than CVS (web browsers are more common than CVS clients, allowing documents to be manipulated with ease on all net-enabled Windows machines with MS Word) and provides roughly the same functionality with binary files. Finding the latest version of a file requires more manual navigation in the forum than in CVS, but this is considered a minor problem.

In short, we think we’ve figured out a decent way to handle version management, so we’ll stick with it to the end of the project if nothing unexpected turns up.

6.4 After end of implementation 3 phase

Use of CVS for code version management and the group’s discussion forum has continued in the same way as at the end of the second implementation phase. Generally speaking, this form of version management has now become routine for the group.

The most severe incident occurred on the 25th of February, when we were suddenly faced with an inability to update from or commit to the CVS. The only clue was the following message:

```
Assertion failed: ptr >= rcsbuf_buffer && ptr <
rcsbuf_buffer +
rcsbuf_buffer_size, file rcs.c, line 1095
cvs [server aborted]: received abort signal
```

After roughly an hour of examining the problem, we determined that this was due to an out-of-date version of CVS running on several HUT-CC servers, including kosh. We notified HUT-CC about this, and the offending CVS installations were updated overnight. Thus, in the morning, everything was fine again.

Also, one of the developers complained to me about the following error message:

```
cvs server: Up-to-date check failed for `IOHandler.java'
```

This message appears if one is attempting to commit a file that has been changed in the repository since one last updated one's local copy. To most CVS users that commit without checking for updates first this message is quite well known. Apparently, this particular developer has not experienced anyone else editing code he was working on before. After this incident, he started locking files in order to work on them without being disturbed, which prevents concurrent development of those parts of the system. Apparently, there is still room for improvement in our management of concurrent development.

A few problems were caused by adding test data to the CVS repository. This made sense while the save functionality was missing, but as soon as we started saving modified test data back to disk, people started getting error messages about missing data files that were referred to by files in the CVS. Test output should not be stored to CVS, so this problem can be solved in one of the following ways:

- By putting the test data to be shared among the users in a different directory than the data directory used by the ProConf system and requiring users to explicitly copy this to their ProConf data directory. Restructuring the directory structure at this time seems disproportionately complicated to me.
- By locking the test data using CVS locking facilities. Unfortunately, this means that attempting to commit changes to the main ProConf directory will cause an error if the test data has been modified.
- By simply not committing test data to CVS. This only requires warning everyone of the consequences. This is the solution that has been implemented.

6.5 After end of delivery phase

During the delivery phase, nothing unexpected happened to the CVS repository. The test data in the repository has stayed usable despite the lack of safeguards. All in all, using the CVS is now routine for all the developers. Similarly, documents appear on the discussion forum when they are ready, with no problems. In short, we have nothing to report about version management during this phase, but, as they say, no news is good news.

7 Overview

Version management in this project has proceeded separately for documents and code. While the code has been in the CVS repository from the beginning, the documents were in the beginning distributed in several different ways: the SoberIT NT file server in TA218, email and the group's discussion forum. By the end of the second implementation phase, only the forum remained in use for document distribution.

Both CVS and the group's discussion forum have been adequate for our needs. A discussion forum is not, however, the optimal solution for version management on its own. Word's change tracking could probably have been used to track changes more effectively, as the current system makes it hard to see what has changed between versions. Some sort of equivalent to CVS with the ability to manage concurrent changes to Word documents would have been ideal, but we are not aware of any such tool.

CVS has been very useful to us, as it has provided a simple way to ensure that everyone receives all updates to the code as soon as possible. It also resolves most concurrent updates automatically, which means that we have spent very little time merging updates. CVS has a few weaknesses: indentation changes are enough for CVS to consider the entire line changed, although its meaning has not changed. Because of this, a few files have, from CVS' point of view, been rewritten completely several times, although only a line or two has really changed. In practice, this did not cause any problems. Also, a few minor technical problems developed, but, again, none of these affected the project badly.

CVS is probably one of the best and most well known solutions to the problem of code version management, and in my opinion, there is no reason not to use it for such needs in software projects with more than one developer. Using a discussion forum to handle documents does not work well with documents that may receive concurrent edits. Therefore, it would be better to use a word processing system capable of saving files that CVS can merge changes in and store the documents in the CVS. Possible formats include LaTeX, HTML, plain text and OpenOffice XML.

---

[1] I. Haikala, J. Märijärvi: Ohjelmistotuotanto, Suomen Atk-kustannus 1998.
[2] SoberIT: A218 computer class,
http://www.soberit.hut.fi/T-76.115/02-03/ohjeet/oht-luokka.html
3 WinCVS,
http://www.wincvs.org/
4 P. Cederqvist et al.: Version Management with CVS,
http://www.cvshome.org/docs/manual/
[5] Kenneth Haglund/PMoC: Communication and meeting practices