

Paper 014-31

Version Control on the Cheap. A User-Friendly, Cost-Effective Revision Control System for SAS®

Tim Williams, PRA International, Charlottesville, VA

ABSTRACT

Revision control is an integral component of code development in many languages but is often absent from SAS programming. Cost and ease of use are often cited as two of the primary reasons why a revision control system is not used. However, the need for such a system is growing. An audit trail of changes to code is becoming increasingly important in many industries. From a programmer's perspective there are many advantages to a system that can easily differentiate between programs that are under development, have been validated, or have been released into a production environment. The ability to roll back changes during development and easily compare revisions to files can greatly increase programming efficiency.

This paper describes the implementation of free, open source components that comprise an easy to use revision control system for SAS programs. The system is deployed in a large organization where SAS is used for the analysis of clinical trials data. The following discussion is intended for programmers at any experience level who want to better manage their SAS programming projects.

Keywords : version control, code development, programming, project management, Concurrent Versions System, CVS, open source, General Public License, freeware

INTRODUCTION

Revision control of source code should be a vital component in any code development project.

"Programming (at any level, small or large) without source control is like playing in the high trapeze without a net...you can do it, and if you're really good you'll get away with it for a while, but when you miss you'll be sorry." (Liebman, 2003)

The safety net analogy is a key concept in source control. How often have you spent time trying to reverse a seemingly innocent change that caused your program to fail? Have you ever deleted a file by mistake, only to wait hours to obtain a copy from backup media - if a backup even exists?

Source control provides far more than just a safety net. An audit trail of code history is produced for the entire lifespan of a project. You can easily identify the status of files and label code that was used to produce a set of output. Systems like the Concurrent Versions System (CVS) allow simultaneous development in multiple work areas with all modifications merged into a single, secure repository.

Other benefits of source control are more subtle. Code management provides the opportunity to implement programming standards. The system may enforce certain rules for program header comments, automatically insert versioning information into the file, and even run code formatting (beautifier) scripts.

With all of these advantages, why is source control not implemented more widely by SAS programmers? Many have simply not been introduced to the methodology. Other important factors include the high cost of commercial source control solutions and the historical lack of user-friendly interfaces in open-source systems. CVS is a free application that has been available for many years as a command line interface. This has hampered its acceptance by users who are more accustomed to a graphical user interface (GUI). Licensing for commercial GUI-based systems can be cost prohibitive, especially for large organizations. Fortunately, free, user-friendly solutions are now available.

GENERAL CONCEPTS

Many authors use the terms *revisions* and *versions* synonymously, but I will reserve the former to mean the incremental changes to a file during its development, and the latter to identify a collection of files (all with varying numbers of revisions) at specific points in a project's lifespan. Code that is run for a specific *release* of the project will be referred to as the *version* of the code used at that time. *Source control*, *revision control*, and *version control* all refer to managing the change history of files.

In source control systems, a *repository* acts as a centralized code warehouse where files, their change histories, and other administrative files are stored. A repository may be housed on a local workstation or remote server accessed over a network. Repositories offer an additional layer of protection by storing code in a secure location that you do not access directly. Placing repositories on servers facilitates centralized management of configuration, backup, and security and allows access by programming teams who are geographically remote from each other.

Repositories should store all the program source code that is needed to recreate the entire set of output for a project. This includes not just SAS program code but also compiled, validated macro catalogs and format catalogs (unless the catalogs are regenerated at run time from other managed programs). Any other metadata essential to running the programs should also be placed under source control. For example, Excel™ spreadsheets may be used to store information about programs or store titles and footnotes for output. Output files and program logs should not be placed under source control. As a general rule, if a file can be reconstructed from other files already under management, then that file should not be placed in the repository. Files generated from source code can easily get out of step when code is changed but not run or code is run but the generated file is not committed to the repository. SAS data sets are managed outside of CVS. These binary files are too large and impractical to store within the CVS repository.

A *sandbox* is a location that contains the working copies of files. You edit and test code in this location after obtaining it from the repository. Separate folders should be used to isolate code *development* from the *production* area. The *cvs checkout* command is used to create a sandbox by checking out files from the repository. After code is edited, the changes are sent to the repository using the *cvs commit* command. A *commit* creates a change history log by recording the username, timestamp, and comments entered at the time you commit the file. The complete change history for a file is available by reviewing the changes associated with each commit. For those of us in the pharmaceutical industry, the Food and Drug Administration's Code of Federal Regulations (21 CFR Part 11) speaks to the use of audit trails for electronic data, "Use of secure, computer-generated, time-stamped audit trails to independently record the date and time of operator entries and actions that create, modify or delete electronic records." (FDA 1997) CVS provides similar audit trail information for the change history of our SAS programs.

Branches and *tags* are labels used to mark a specific revision of a file, group of files, or an entire project at specific points in time. These labeling concepts are described in greater detail later in this paper.

COMPONENTS OF THE REVISION CONTROL SYSTEM

We have adopted a modular approach to our validation and implementation. If one component is found to lack the required functionality, we can replace it with other software. This modular approach affords us flexibility as our requirements evolve.

Our revision control system is comprised of three modules:

- CVS-NT - the core CVS service
- Tortoise CVS - a graphical user interface
- ExamDiff - a utility for comparing revisions

CONCURRENT VERSIONS SYSTEM (CVS-NT)

CVS started out as a collection of shell scripts posted by Dick Grune to the comp.sources.unix newsgroup in December of 1986. While none of the original code from these scripts remains in the current version of CVS-NT, it formed the foundation for what would become one of the most widely-used revision control systems. In 1999 Tony Hoyle converted Unix CVS to run under the Windows operating system as CVS-NT. CVS-NT is free, open source software licensed under the GNU General Public License (GNU GPL) and available at www.march-hare.com/cvspro.

TORTOISE CVS (TCVS)

Say goodbye to the command line interface.¹ Graphical User Interfaces (GUIs) for CVS-NT are now available that make revision control an enjoyable experience even for novice users. Like CVS-NT, TCVS is also available free of charge under the GNU GPL. It was originally based on the source code for another GPL GUI client for CVS named WinCVS.

TCVS is installed as a Windows shell extension and provides point-and-click access to CVS commands as drop-down menus within Windows Explorer™. You simply right click on files and folders to access context-sensitive CVS menus (**Figure 1**).

In addition to a point-and-click interface for the most common CVS commands, TCVS allows you to quickly identify the status of files in a sandbox. Icon overlays of various images and colors allow you to determine the status of each file (up-to-date, requires a commit, not managed by CVS, etc.).

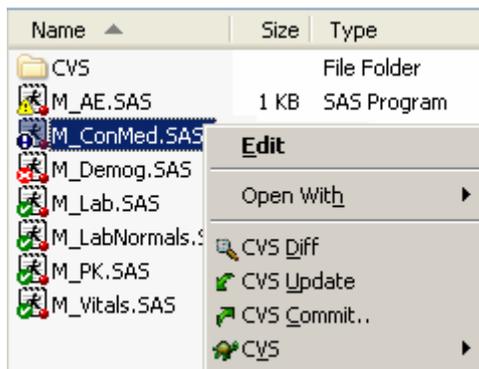


Figure 1 : TCVS commands within Windows Explore

File revision history is available in a window that lists time stamp, username, and commit comment for each revision. History may also be displayed in a revision graph (**Figure 2**). Passing the mouse over the revision numbers and labels reveals the time stamps, usernames, and commit comments. The graph is not solely for display purposes. You can select any two revisions and send them into the comparison application, retrieve any revision into the sandbox, or apply labels.

Source code, installation files, and documentation are available at the TCVS home page (www.tortoise cvs.org).

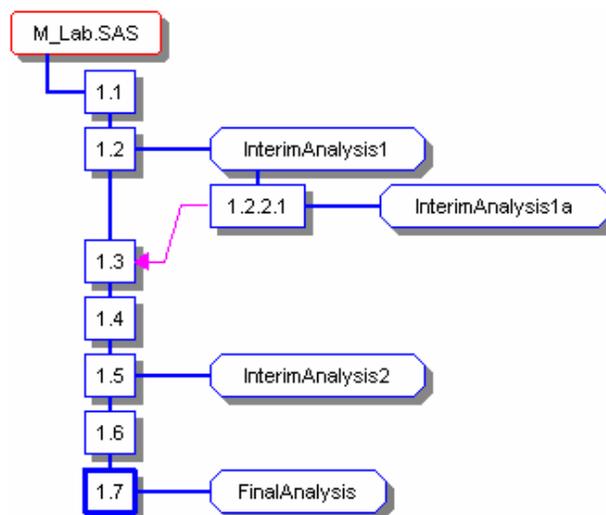


Figure 2 : Revision graph

EXAMDIFF

Any source control system should include the ability to compare differences between revisions. CVS provides this capability through its native `cv diff` command, but external components may be added to enhance this feature.

We use Examdiff as the external diff utility (www.prestosoft.com). The application is available in a freeware version with limited capabilities or a fully functional commercial version. The freeware version provides basic functions:

- integration with TCVS
- simultaneous display of two revisions
- visual identification of changes, including the type of change (added, altered, deleted lines) with easy navigation through the changes
- option to display only lines that differ

¹ Only if you want to. Retro-grouches and advanced users take heart - TCVS includes a complete installation of CVS-NT command line interface.

- option to include/ignore white space in the comparison
- option to include/ignore case in the comparison

In addition to these features, the commercial version identifies exactly what content has changed within lines and allows editing of code within the display. We configured the freeware version to send the displayed program code to the SAS Enhanced Editor for editing (other editors are easily specified).

Alternatives to Examdiff include WinMerge (available at www.sourcforge.net) or the code editor UltraEdit's new companion module UltraCompare (www.ultraedit.com).

FACILITATING CODING STANDARDS

CVS may be used to facilitate code standardization and consistency across programming teams. Among its capabilities are the resolution of keywords within the program code, examination of entire programs for code standardization (including use of code formatting/beautifier scripts), and standard naming conventions for labeling releases.

KEYWORD SUBSTITUTION

When CVS finds certain keywords in files it inserts values that provide information about that specific revision. Keywords take the form `$Keyword$` and can be used to identify the file (`$RCSfile$`)²; revision number (`$Revision$`); username of the person who committed the revision (`$Author$`); time and date the revision was last committed (`$Date$`); detailed header information that includes date, author, revision number, path to file (`$Header$`); and commit messages (`Log`). A complete list of available keywords can be found in most sets of CVS documentation (Cederqvist, 2003; Thomas and Hunt, 2003; Vesperman, 2003).

Novice users may be tempted to include all available keywords in their program headers, but some keywords can be problematic (see Vesperman, 2003 for issues not mentioned here, including how keywords may complicate merging and branching). The `Log` keyword appears attractive since it provides a detailed account of the program's development history (Peterson and Cherny, 2000). However, its use is strongly discouraged for several reasons. Unlike most keywords, the log values are not overwritten with new values when a file is committed - they are appended to the list already within the program. The `Log` field may grow to substantial length and include comments that may be inappropriate to view outside of the development team. There is also a strong temptation for programmers to edit the log comments within the file itself - simply correcting spelling errors or adding to the comments, as one would with any normal program header information. Such actions cause the comments to differ from those in the CVS system, leading to doubt in the accuracy of the file's change history. The log information is easily accessed within CVS, so it is best for it to remain in that secure location.

Bearing this information in mind, we chose to use only the `$RCSfile$`, `$Revision$`, `$Author$`, and `$Date$` keywords as part of our standard SAS program header. An abbreviated example of the header prior to committing the file appears as:

```

/*****
PROGRAM NAME   : $RCSfile$
REV/REV AUTH  : $Revision$ $Author$
REV DATE      : $Date$
** - - - - DO NOT EDIT ABOVE THIS LINE - - - - **
CLIENT/PROJECT: DrugCo/DC001
PURPOSE       : Derivation of laboratory data
ORIG AUTHOR   : Tim Williams
... etc.
*****/

```

² The `$RCSfile$` keyword resolves to the name of the file within the repository. Files in the repository are stored in RCS format with the file name appended with a `,v`. For text files, the RCS format includes the initial revision entered into the repository and all subsequent differences between revisions.

The keywords resolve to their values³ when the file is committed to the repository:

```

/*****
PROGRAM NAME   : $RCSfile: M_Lab.sas,v $
REV/REV AUTH   : $Revision:1.1 $ $Author: WilliamsTim $
REV DATE      : $Date: 2004/01/20 16:27:11 $
** - - - - DO NOT EDIT ABOVE THIS LINE - - - - **
CLIENT/PROJECT: DrugCo/DC001
PURPOSE       : Derivation of laboratory data
ORIG AUTHOR   : Tim Williams
... etc.
*****/

```

The `$Revision$` keyword provides a link to the revision number that was recorded on our quality control checklists when the program was assessed. Our programmers, clients, and auditors can quickly determine if the program has been QC'd by cross referencing the revision number in the program header with the number recorded on the corresponding checklists. This replaces error-prone methods of manually entering revision numbers in program headers or relying on file system dates as surrogates for revision numbers.

SCRIPTING FILES

CVS can call accessory scripts when files are committed, labeled, or changed in other ways. Scripts may be setup to run on only specific projects, folders, or files. Some of the scripting files we use are described below.

commitinfo

This file defines scripts to run when files are committed to the repository, before the actual commit takes place. Parameters passed to the scripts include the full path of the repository and the names of all files being committed. If a script called by `commitinfo` fails on any of its conditions, the commit of the file does not take place and you receive an error message detailing possible corrective action. As for all scripts described in this paper, the error message is configured within the script itself and can be customized to suit your requirements.

In our implementation, `commitinfo` calls a single Perl script that performs the following:

1. Allows only SAS programs and compiled catalogs to be committed to the repository.
2. Generates warnings for unnecessary spaces or tabs at the ends of lines. Changes in the amount of empty space on line endings may cause CVS to report that contents of a file have changed when the code content was not altered.
3. Scans SAS programs for the required CVS keywords and fails to commit the file if they are not present. A message box informs you which keywords are missing from the program header.

Since the scripts called from `commitinfo` can scan entire files, they may be used to check that programs conform to coding standards. Programs that fail the checks will not be committed unless the discrepancy is resolved. Issues that are not critical can pass warning messages back to you that identify the offending lines in the program, but still allow the commit to proceed.

verifymsg

The file `verifymsg` specifies scripts to run immediately after a log message is entered for a commit, but prior to the execution of the commit. The log message is passed to the script which may then examine, parse, or modify it as needed. The script may be used to ensure that messages entered during the commit follow project standards and contain required information. The `verifymsg` file is often used in conjunction with another CVS file, `rcsinfo`, that defines templates for entering specific information during a commit.

We use a Perl script called from `verifymsg` to determine if the log message adequately describes the changes made to the file. If you omit a log message or enter non-descriptive text (for example, "None" or "No Message"), then the commit fails and you are informed that you must enter text that describes the changes you made to the program.

taginfo

`Taginfo` specifies scripts to run before branch or tag labels are applied. We use this file to call a Perl script that enforces a naming convention and prevents deleting or moving labels. This is important in a highly regulated

³ We use a keyword resolution mode that places both the keyword and its value in the file. See the section on the `commitinfo` scripting file for how this mode is used to check that keywords exist in the program header.

environment. We use labels to identify revisions of programs that were run to produce output sent to our clients or submitted to the FDA.

BRANCH AND TAG NAMES

CVS allows you to label a file, group of files, or an entire project. When teams work on multiple projects it is important to develop a standard naming convention that identifies key events. Labels allow you to return to the exact programs that were used to produce output at specific stages of a project.

We developed a naming convention that identifies the type of event, a sequence number (if applicable), and a sequence letter:

```
InterimAnalysis<n><a>
FinalAnalysis<a>
```

where: <n> denotes the nth occurrence of that event type (first interim analysis as InterimAnalysis1 and second interim analysis as InterimAnalysis2)

<a> is a letter used to identify versions within that event (when corrections are made to code that has already been labeled using a branch name)

It is important to differentiate between tags and branches. As the name implies, branches allow you to isolate development onto a separate branch. We use tags to identify events on branches because branching on branches quickly becomes confusing. These concepts are best illustrated with an example found in the next section.

AN EXAMPLE OF WORKING IN THE CLINICAL TRIALS PROJECT TIMELINE: BRANCHING, TAGGING AND MERGING

Figure 3 illustrates how you may use branching, tagging, and merging as project proceeds from the start of programming through two interim analyses and a final analysis. While this example relates to the pharmaceutical industry the concepts may be applied to any project.

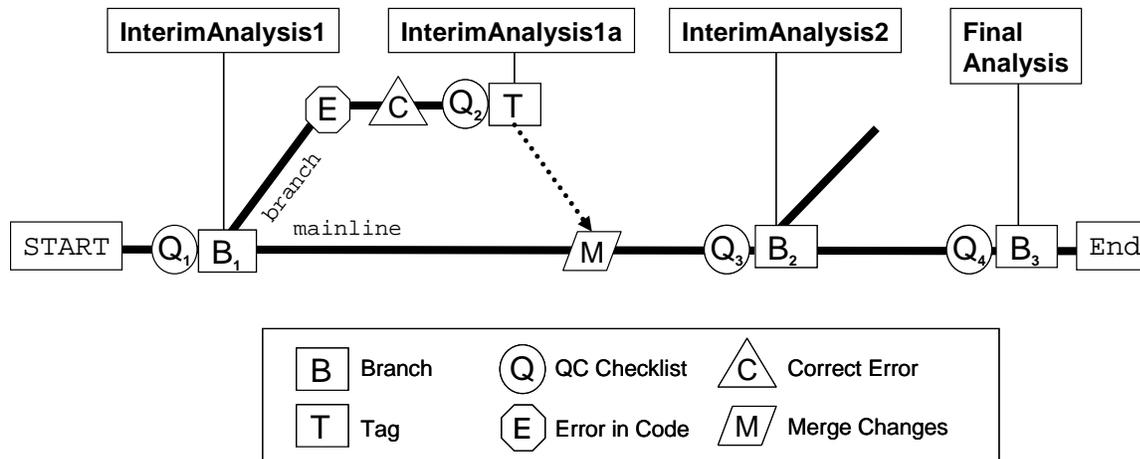


Figure 3 : Branching, tagging and merging in a clinical trials project timeline

You start by programming toward your first objective, an interim analysis of clinical trials data. After all programs are written they are evaluated using a quality control checklist (Q₁). Once QC is complete, your files are identified as production-ready by applying a branch name (B₁). In this example the branch name “InterimAnalysis1” is applied to all the SAS code used to produce output for this event. The code is then moved to a production area and run to produce the desired output.

After output for the first interim analysis is completed, you commence working toward the next objective which is the second interim analysis. You change, add, and delete code and programs as you work toward your next goal. Then you discover a small error is present in one of the programs used in the first Interim Analysis (E) that must be

corrected and then applied to all remaining deliveries. You also want to make small changes to a number of other programs. You have already spent many hours programming toward your next objective. Is all this work for nothing? Thanks to CVS, it is not.

You inform your programming team about the required corrections and changes. Timelines are tight, so the rest of the team keeps working on the mainline of development (working toward the second interim analysis). You are left to sort out the changes and re-run the first interim analysis. You checkout the project files using the tag "InterimAnalysis1". Now you have a copy of all the programs as they were at that point in time. You correct the error and make the other changes (**C**) on the branch. The rest of your team is progressing steadily toward their next goal. This is referred to as *parallel development*. With your changes complete and QC'd by a second programmer, you apply the tag "**InterimAnalysis1a**" and move the files into the production area to produce corrected output for the first interim analysis.

Once the corrected interim analysis output is complete you are free to rejoin development back on the mainline. CVS can automatically merge your changes on the branch back into the mainline of development, ensuring the changes made for the first interim analysis are also reflected in all subsequent outputs. You call a halt to all development on the project and each programmer commits their work. You now use CVS to merge the changes from the **InterimAnalysis1a** branch to the mainline (**M**). CVS will identify changes in the branch that conflict with changes in the mainline and allow you to resolve them. It is your responsibility to resolve any conflicts since CVS can not judge which change is intended or correct.

Development then proceeds smoothly toward your next goal. Programs that changed after the first interim analysis are evaluated using the quality control checklists (**Q₃**) and the code is assigned a branch name (**B₂**) of "**InterimAnalysis2**". Fortunately, no rework of the second interim analysis was needed, but a branch provides the opportunity just in case. Work then continues toward to the final analysis and project closure.

ENHANCING PRODUCTIVITY

Some of the greatest gains in productivity occur during the day-to-day use of a revision control system. You may have experienced the agony of making a seemingly minor change to a program only to have the program fail when it is run. What then follows is the laborious task of trying to retract those changes in an effort to return to code that runs without error. CVS takes the guess work out of this process by quickly allowing you to identify what has changed and to easily roll back to an earlier revision.

If you accidentally delete a program it may take a considerable amount of time to restore it from backup media (if you even have a backup copy at all). In large organizations you may need to involve support staff to obtain the backup copy. However, with CVS you simply right-click on the parent folder and select the *cv*s *update* command from the menu to retrieve the most recent copy of the file from the repository.

Our clients increasingly request that our SAS programs are validated and run in a production environment. In the past, this meant manually moving programs from a development area to a production area after the programs passed a quality assessment. This resulted in copies of programs in multiple locations as production runs are completed and archived during the project life cycle. Use of CVS replaces the need for multiple copies. The repository identifies snapshots of a project internally, without the need for external copies in multiple folders on the file system. We still separate development from production areas, but moving files into production is as simple as a few mouse clicks and selecting the appropriate label.

VALIDATING OPEN SOURCE SOFTWARE

In the pharmaceutical industry, the FDA requires validated computer systems for handling clinical trial data (FDA, 1999). By extension, programs that manipulate data and applications that manage these programs should also be validated. Some may feel that validation of open source software presents challenges beyond what is needed for commercial software. Others counter that since the code and development process for open source software is visible to all users and multiple developers, these applications are more robust than commercial (closed) systems. Regardless of your view and whether or not your industry mandates validation, it is always a good idea to evaluate the software in the context of your own environment.

SAS Software is a suite of generally accepted applications in the pharmaceutical industry and we can refer to papers like "The Quality Imperative - SAS' Commitment to Quality" (SAS Institute, 2003) as evidence of its support and validation. At PRA, we additionally validate The SAS System using a Systems Development Life Cycle approach (Walsh and Johnson, 2001).

Validation of our implementation of CVS, TCVS, and ExamDiff also followed the Systems Development Life Cycle approach. All components progressed through Concept, System Quality Assurance, Systems Requirements Specification, and Acceptance Testing phases. Detailed testing of the system functionality, configuration, and auxiliary scripts was completed and summarized in a Validation Report. A System Configuration Management Plan details how changes to the system will be evaluated and implemented in a controlled manner. Installation and training plans complete the set of documentation that serves as an important resource within the company and for auditors who wish to review the system.

THE COSTS OF *FREE* SOFTWARE

“You’d be surprised how much it costs to look this cheap.”

- Dolly Parton

Before you start making a pitch for a solution that won’t cost a dime, consider some of the related expenses.

Hardware. You should have a server that is dedicated to running the CVS repository. In a pinch, CVS repositories can coexist on a server with other applications (even your SAS server), but this is not recommended by the CVS-NT development team. In global organizations you may require multiple servers spread out across the organization to serve multiple programming teams.

Software. Yes, there may be software expenses. For CVS-NT you will need to budget for the client and server operating systems. You may also choose to augment your system with commercial products or develop custom software in-house for additional functionality.

Training and Implementation. The learning curve associated with any new system causes a period of decreased productivity as you adjust to the new methodology. Develop a training program that allows hands-on experience to help lessen the impact and improve user acceptance. Early use by a select group of motivated users builds a ground swell of support from the grass roots. Anticipate delay periods as you work to overcome the misconception that free software is somehow inferior or resistance from staff who are reluctant to implement systems that change existing processes.

Maintenance and Support. You must budget time for setup and maintenance of the repository server, importing new projects, training, and support of the user population. The amount of time spent on these tasks is not trivial and must be considered in addition to preexisting commitments and costs. You may also wish to budget for commercial support of the system, though at PRA we manage all support internally.

Most of these issues are common to any revision control system, whether commercial or free. When the high costs of commercial source control solutions are added to the items above, the free alternative becomes all the more attractive.

THE MYTH OF NO SUPPORT

Companies may have concerns about a perceived lack of available support for open source software. When considering an open source application it is important to investigate the amount of active development that is occurring, the scope of the user base, and support from the internet community. Both CVS-NT and TCVS are well established projects, contain an active development community (www.sourceforge.net), and receive strong support from a diverse group of users.

The March Hare company now offers support for CVS-NT (www.march-hare.com).

ALTERNATIVES

Many free and commercial source control solutions are available. Your choice may be determined by a combination of funding, operating system, and philosophy.

Some of the more popular commercial systems include Visual Source Safe (msdn.microsoft.com/ssafe/), ClearCase (www.rational.com/products/clearcase/index.jsp), BitKeeper (www.bitkeeper.com), and Perforce (www.perforce.com) to name a few. SAS Drug Development (www.sas.com/industry/pharma/develop/) contains version control functionality for SAS code, along with a host of other features for FDA compliant programming and data storage.

A system called Subversion (subversion.tigris.org) is emerging as a rival and potential replacement for CVS. Subversion was started in 2000 by a group of developers who desired a system similar to CVS, but without its bugs and limitations. The developers of TortoiseCVS are actively working on a Windows Explorer shell extension called TortoiseSVN (tortoisesvn.tigris.org/) that will provide an interface to Subversion, as TCVS does for CVS.

LIMITATIONS

Use of a source control system does not imply that the managed code is in any way superior to code not under management. Source control facilitates close linkages between code and quality control checklists and provides rudimentary audit trails, but these are only as good as the processes that surround them. Use of CVS must fit into a larger process that includes programming standards and validation methodologies. CVS is also not a replacement for communication within your programming teams.

CVS itself has some technical limitations that warrant discussion. Renaming files is difficult, though it should become easier in future releases. Current work-arounds may result in loss of a file's change history or revision numbering. Proper training and involvement by the CVS Administrator can prevent loss of this information. Even though this approach is more complex, the frequency of renaming is so low that we currently manage it this way to ensure maintenance of the audit trail. Renaming or deletion of folders (directories) is also problematic, but these events are rare in a well managed programming environment.

Branch and tag names can not be time stamped or altered automatically by the system. These labels can only be evaluated in a pass/fail context to determine if the labels fit naming conventions. Unless additional scripts are used it is impossible to determine who applied a label and when it was attached to a revision.

TCVS provides a convenient user interface but advanced users will find themselves occasionally relying on the CVS command line for advanced tasks. Although WinCVS provides a more in-depth feature set, it lacks the simple and efficient integration with Windows Explorer.

CVS is not integrated with an electronic signature system so the change history for program code can only be considered a rudimentary audit trail. Depending on your needs, you may wish to consider more complex and comprehensive applications such as SAS Drug Development[®].

CONCLUSION

At PRA we are successfully managing over 160 client projects on six repository servers in North America and Europe. The system has proven itself in both development and production environments. Current challenges include the continuing need to train new users and provide refresher courses for existing staff.

We are considering the replacement of our current diff utility with an open source alternative called WinMerge that integrates more closely with CVS, allows editing of files within the application, and can automatically merge differences in files (unlike the freeware version of ExamDiff). We are also considering the addition of an open source application called CVSMailer that can be configured to send email notification to team members for CVS events (Berglund, 2004). We continue to watch the development of the Subversion system with interest but have no current plans to replace CVSNT.

The security, features, and efficiencies provided by source control provide our clients with a better product at lower cost.

REFERENCES AND RESOURCES

Berglund, Bo. 2005. CVSMailer. <<http://web.telia.com/~u86216121/cvsmailer/CVSMailer.html>> (December 27, 2005).

Cederqvist, Per et al. 2005. *Version Management with CVS*. Free Software Foundation, Inc. Available in stable release and WIKI formats. <<http://ximbiot.com/cvs/manual/>> (December 27, 2005).

CVS-NT. <<http://www.march-hare.com/cvspro/>> (December 27, 2005).

Fogel, Karl. 2000. *Open Source Development with CVS*. Published under the GNU GPL by the Free Software Foundation and available at: <<http://cvsbook.red-bean.com/>> (December 27, 2005).

Free Software Foundation. 1991. *GNU General Public License, Version 2*.

<<http://www.gnu.org/licenses/licenses.html#GPL>> (December 27, 2005).

Liebman, Roberto. 2003. Comments in the Slashdot Book Reviews forum for *Pragmatic Version Control Using CVS*. <<http://www slashdot.org>> (December 30, 2003).

March Hare Pty Ltd. Commercial support for CVSNT. <<http://www.march-hare.com/index.htm>> (December 27, 2005).

PrestoSoft 2005. ExamDiff - Visual File Comparison Tool. <http://www.prestosoft.com/ps.asp?page=edp_examdiff> (December 27, 2005).

Purdy, Gregor. 2000. *CVS Pocket Reference*. O'Reilly & Associates, Inc, Sebastopol, CA.

SAS Institute Inc. 2005. *"The Quality Imperative - SAS' Commitment to Quality"*. <<http://www.sas.com/whitepapers/index.html>> (December 27, 2005).

Thomas, David and Hunt, Andrew. 2003. *Pragmatic Version Control Using CVS*. The Pragmatic Bookshelf. Raleigh, NC. <<http://www.pragmaticprogrammer.com>> (December 27, 2005).

Tortoise CVS. <<http://www.tortoiseCVS.org>> (December 27, 2005).

US Food and Drug Administration. 2003. *Guidance for Industry. Part 11, Electronic Records; Electronic Signatures - Scope and Application*. <<http://www.fda.gov/cder/guidance/5667fnl.htm>> (December 27, 2005).

US Food and Drug Administration. 1999. *Guidance for Industry, Computerized Systems used in Clinical Trials*. <http://www.fda.gov/ora/compliance_ref/bimo/ffinalcct.htm> (December 27, 2005).

US Food and Drug Administration. 1997. *21 CFR Part 11, Electronic Records; Electronic Signatures; Final Rule*. Federal Register Vol. 62, No. 54, 13429. <http://www.fda.gov/ora/compliance_ref/part11/frs/background/11cfr-fr.htm> (December 27, 2005).

Vesperman, Jennifer. 2003. *Essential CVS*. O'Reilly & Associates, Inc, Sebastopol, CA.

Walsh, Bucky and Johnson, Greg. 2001. *Validation: Never an Endpoint: A Systems Development Life Cycle Approach to Good Clinical Practice*. Drug Information Journal Vol. 35 pp. 809-817.

WinCVS. <<http://www.wincvs.org>> (December 27, 2005).

ACKNOWLEDGMENTS

I wish to thank the SAS programmers at PRA International for their continuing comments and assistance.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. The author may be contacted at:

Tim Williams
SAS Systems Administrator
PRA International
4105 Lewis and Clark Drive
Charlottesville, VA 22911
Email: WilliamsTim@PRAIntl.com
Web: www.praintl.com

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.