JOIN OVER 100,000 PROS! Join Log In | Member Benefits

STICKYMINDS

Agile Testing DevOps Security AI

Evolving beyond Version Control for Agile Development

ARTICLE

By Steve Berczuk , Brad Appleton Robert Cowham | May 30, 2006

SUMMARY

This article looks at satisfying the principles of Configuration Management with varying degrees of tool support, which we classify from Version Control at one end, to true CM tools (and beyond!). Many agile developers restrict themselves to tools that are more on the version control end, so we aim to highlight the differences, and particularly point out certain features which can make your life a lot easier.

We will use the CMMI (Capability Maturity Model Integrated) definition of Configuration Management:

The purpose of Software Configuration Management is to establish and maintain the integrity of the products of the software project throughout the project's software life cycle. Software Configuration Management involves identifying configuration items for the software project, controlling these configuration items and changes to them, and recording and reporting status and change activity for these configuration items.

At its heart, CM is intended to eliminate the confusion and error brought about by the existence of different versions of artifacts. Artifact change is a fact of life: plan for it or plan to be overwhelmed by it. Changes are made to correct errors, provide enhancements, or simply reflect the evolutionary refinement of product definition. CM is about keeping the inevitable change under control.

Can you "do" CM using only Version Control tools—absolutely! But the better you understand the principles, and the better you understand what tools can do for you and how they have evolved, particularly over the last 10 years or so, the easier you can make your life.

Most Popular Tools

Sometimes we're asked what are the "top three version control tools today." People expect us to answer what our personal top three favourites are—instead we tell them

We use cookies to ensure you get the best experience on our website.

Accept

Read More

#3: CVS—because it is "free", and comes "out of the box" on most Unix/Linux systems (note this is slowly evolving towards Subversion)

#2: VSS (Visual SourceSafe)—because it comes as "part of the package" with many purchases of Microsoft's Visual Studio IDE

and the **#1** version control tool in use today is ...

"Version control? What's that mean? Isn't that what backups are for? I already do that!"

While this may seem apocryphal, one of us very recently came across an organisation where some teams are still using just the file system to do version control—manually copying changed files between directories, etc.

History of Version Control and Configuration Management
Principles of Configuration Management have been understood throughout civilisation, particularly in the military arena. A thread on CMCrossroads points out the discovery of timbers from ships from the time of Nelson and Napoleon showing identification of configuration items and an understanding of management of configurations which was perhaps one contributing factor to Britannia ruling the waves in that era!

The OS/360 development team made famous by Fred Brooks' "Mythical Man Month" and working in the 60's, did configuration management but without direct tool support. They used what they called a "vault" system, which was simply a set of "backed-up" storage areas (integration workspaces) for each "promotion level" they needed. They had a development "vault" (an integration workspace for development), a more formal "vault" for what they would periodically release to integration+test, and finally a "release" or "production vault" for what they actually shipped.

The first widely used tool for version control was **SCCS**, originally written by Marc Rochkind in 1972. The simple model of being able to check-out and check-in, saving versions and showing differences between them, was a big advance on previous manual methods.

Implementing CM
So the first question is: can we do CM using a VC tool? Since it is possible to do CM without any tool support at all, the obvious answer is yes, although it will be rather painful and a very manual process. Let's consider some of the features of tools along the VC to CM continuum.

Tool Features for VC and CM
We suggest the following classification of tool features:

- Basic Version Control (Library Management)
  - Storage of deltas (archival)

We use cookies to ensure you get the best experience on our website.

- Intermediate VC (Concurrent Development)
  - Client-server architecture (not definitive)
  - File-based Branching with simple merging
  - (Example tools include CVS)
- Advanced VC (Parallel and Distributed Development)
  - Atomic transactions
  - Project-oriented Branching with Merge Tracking
  - Workspace management
  - Remote and/or Multi-site development
  - (Example tools include Subversion—except for merge tracking)
- Configuration Management
  - Task-based Development (e.g., change-tasks & change-sets)
  - Configuration-Items (e.g., components)
  - Integrated Change/Defect Tracking
  - Workflow/process enforcement (Example tools include majority of serious commercial tools)

Some other features which may also be important include: scalability and cross-platform support, graphical user interfaces, build & release engineering, problem/issue tracking and process (workflow) support.

Architecture

The original VC tools were all client only, less of a problem in the Unix multi-user environment, for example, where people worked on a single machine most of the time. The archive or repository files in which different versions are stored (usually in a delta format), are directly manipulated by the client programs. The main problems with this are:

- *Security*: since all clients require read/write access to the archive files it is difficult to properly secure files and prevent malicious, or inadvertent damage
- *Reliability*: what happens if your client machine dies in the middle of an update? If you were checking in a group of files only some of the check-in may have happened giving an inconsistent logical state. In addition, the archive files themselves may be corrupted
- *Performance*: requiring direct access to files on a network file share usually does not perform as well as client/server

Thus CM tools are typically client/server architecture (although there are a couple of exceptions—see comments on this article).

A related differentiator between VC tools and CM tools is the handling of labels (or their equivalent). This comes back to implementation: a VC tool is very file oriented, and operations on sets of files (whole configurations of the system) are implemented via updates to lots of individual VC archive files. Thus labelling becomes progressively slower as the number of files and versions grows. A CM tool will tend to store label information in a database, making the act of labelling a constant time operation. This

We use cookies to ensure you get the best
experience on our website.

more resistant to using the tool or doing the action, and will be tempted to find work-arounds or shortcuts (thus defeating the principles).

Considering open source tools, CVS stores labels in all the individual files that form the repository. Subversion recognised the inherent performance problems and this was one of the issues they addressed.

Agile Approaches to CM Tools

One of the approaches found in agile practices such as XP is "do the simplest thing that could possible work". The interesting thing is that some agile developers also take this approach to tool usage—keep it simple.

To this end, there are large numbers of agile teams using CVS, and fairly happily. In this instance, the agile practices of test driven development with a full set of unit tests available to developers, combine to make it relatively straightforward to check-in on the trunk or mainline and keep the mainline advancing forward. One of the problems with CVS is the lack of atomic transactions, covered in the next section.

Atomic Commits, Change-sets, Configurations and Streams

These days, this feature is a pre-requisite for all serious tools, although some major players still do not offer it out-of-the-box. It has some major benefits for the evolution of configurations as well as simply for safety and reliability. Regarding reliability, atomic transactions and change-sets avoid the problem with partial check-ins mentioned above that tends to occur with a non client/server architecture (while theoretically possible to offer atomic transactions with a non client/server implementation, in practice this is very difficult).

Frank Schophuizen **wrote in his blog** about his definition of a stream:

- *Streams:* evolve configurations from their current state to a new state closer to the final state (the product release state)

The VC view of a configuration is that it is a set of files and particular revisions of those files (usually represented as a label). Obviously, in the abstract, as the number of files and revisions in your repository mounts up, the number of different possible configurations explodes. Using the Stream concept, you reduce the complexity of managing your configurations by considering the relationship to time (as well as quality etc). Change-sets reduce the complexity dramatically if people are checking in consistent change-sets that are related to a single logical task for a single change (if the change-sets are cohesive, i.e. you don't fix more than one bug in a single change-set unless the bugs really are related). Change-sets group changes together and provide a higher level unit of change to the repository which is much easier to track. If you can restricting your potentially interesting configurations to just the change-sets, then the complexity is much

We use cookies to ensure you get the best experience on our website.

Thus in a day, 10 developers may have checked in 50 change-sets, with a total of 500 individual versions of files. If you are able to link your change-sets together and associate them with specific tasks that required those changes (as described in Austin Hastings' article about **Task Level Commit**), then you can reduce the number of interesting configurations to perhaps 20 tasks completed. This is a significant step. As noted in the comments below, Subversion currently does not offer the ability to link tasks to the change-sets that implement the tasks—this reduces traceability and makes things like release note production a more manual process.

A potential problem is when Developer A checks-in a change set 123 and Developer B checks-in 124 which shares some of the same files in 123. If B just checks-in his change without taking account of A's change, then the new configuration will be "broken" (and thus have less value).

Development practices such as test driven development with a full set of unit tests available to developers, combine extremely well with change-sets, to keep the number of configurations manageable. For example, if you make a change to some files in your workspace including the tests, and you don't check-in until your tests work, you are creating a new configuration with some known value (e.g. the number of new tests which now pass). If during the check-in process, you have a conflict with another developer's just checked in changes, then the whole change set fails, and must be fixed. In the above example, the initial attempt to check-in change set 124 would fail, developer B would merge in A's changes, re-run his own tests, ensure they pass and now check-in 124 again which this time works.

When they are available, change-sets are usually implemented as a unique id, often an integer. In many respects they are built in labels or snapshots in time across your entire repository (or across only the set of changed elements, depending upon the tool's implementation). The use of labels though can cut across change-sets, and when used in this manner indiscriminately, greatly increases complexity. For example a label may include all files as of change set 124 except for a couple of files from change set 100 (the version in change set 124 broke something). This is not a problem for any individual label, but it can be the start of a slippery slope to greater complexity.One of the authors had a client with 85,000 labels in their repository (in this case they were applying VC thinking and techniques to their use of a CM tool which offered change-sets and branching). And these weren't just labels, but labels of labels, which gave different results when combined in different orders due to overlaps. Trying to understand which version of which files was in a particular label, or even worse, which labels any particular version of

We use cookies to ensure you get the best experience on our website.

all the tests run for every committed check-in. Atomic transactions and change-sets are a big help towards this goal. And even better is is the linking of tasks to change-sets.

Parallel Development—Branching

VC tools often support a basic level of parallel development via conflict detection and merge on check-in. Other alternatives include exclusive check-out, although that tends to have significant effects on productivity. The branching scheme used by many VC tools does work for individual files (e.g. version 1.2 is branched to 1.2.1.1), but does not scale well to groups of files (even hundreds, let alone thousands or tens of thousands). This is particularly true of merging, and merge tracking—a quite tricky problem to solve well, as evidenced by the fact that Subversion still has merge tracking on its medium term agenda rather than as an implemented facility. We have all seen organisations where some level of parallel development was required, and as a result a complete copy of the repository was taken and development proceeded in its own little "branch". This leads to either divergence or some very painful merge scenarios, and a huge hit on productivity.

Conclusion

We haven't covered all the features of full CM tools, but the most important point is to understand the principles of CM, since you can then apply whatever tools you have to satisfying those principles. The challenge then becomes identifying those aspects of CM that are really important to you, and understanding the different levels of tool support and how they can make processes that were previously an out-of-reach mirage suddenly become attainable. Tool support really can make a big difference, but it comes a poor second to good understanding. Because of the inherent discipline that test driven development involves, there are many agile teams successfully using CVS. Many have switched to Subversion and are benefiting from change-sets and better branching model. But agile developers can seriously benefit further from making sure they are fully aware of what modern tools are capable of, and ensuring that they use best practice in applying those tools during development and maintenance.

Parallel Development—Branching

VC tools often support a basic level of parallel development via conflict detection and merge on check-in. Other alternatives include exclusive check-out, although that tends to have significant effects on productivity. The branching scheme used by many VC tools does work for individual files (e.g. version 1.2 is branched to 1.2.1.1), but does not scale well to groups of files (even hundreds, let alone thousands or tens of thousands). This is particularly true of merging, and merge tracking—a quite tricky problem to solve well, as evidenced by the fact that Subversion still has merge tracking on its medium term agenda rather than as an implemented facility. We have all seen organisations where some level of parallel development was required, and as a result a complete copy of the repository was taken and development proceeded in its own little "branch". This leads to either divergence or some very painful merge scenarios, and a huge hit on productivity.

Conclusion

We use cookies to ensure you get the best experience on our website.

that are really important to you, and understanding the different levels of tool support and how they can make processes that were previously an out-of-reach mirage suddenly become attainable. Tool support really can make a big difference, but it comes a poor second to good understanding. Because of the inherent discipline that test driven development involves, there are many agile teams successfully using CVS. Many have switched to Subversion and are benefiting from change-sets and better branching model. But agile developers can seriously benefit further from making sure they are fully aware of what modern tools are capable of, and ensuring that they use best practice in applying those tools during development and maintenance.

Topics: configuration management, version control,

We use cookies to ensure you get the best experience on our website.

ABOUT THE AUTHOR

Steve Berczuk

Steve Berczuk is a Principal Software Engineer with experience as a manager, Scrum Master and technical lead in Boston, MA. The author of *Software Configuration Management Patterns: Effective Teamwork, Practical Integration*, he is a recognized expert in software configuration management and agile software development. Steve is passionate about helping teams work effectively to produce quality software. He has an M.S. in operations research from Stanford University and an S.B. in Electrical Engineering from MIT, and is a Certified ScrumMaster. Contact Steve at steve@berczuk.com or visit berczuk.com and follow his blog at blog.berczuk.com.

Brad Appleton

Brad Appleton is an Enterprise Agile+DevOps leader, coach & manager, and seasoned DevOps/ALM/CM solution architect at a large Fortune 100 company. Currently he helps organizations and teams scale, adopt and apply lean/agile development methods and DevOps/ALM/CM practices and tools. He is co-author of *Software Configuration Management Patterns*, the Streamed-Lines branching patterns, and a former section editor for *The C++ Report*. You can read Brad's blog at blog.bradapp.net.

Robert Cowham

Robert Cowham has long been interested in software configuration management while retaining the attitude of a generalist with experience and skills in many aspects of software development. A regular presenter at conferences, he authored the Agile SCM column within the CM Journal together with Brad Appleton and Steve Berczuk. His day job is as Services Director for Square Mile Systems whose main focus is on skills and techniques for infrastructure configuration management and DCIM (Data Center Infrastructure Management) - applying configuration management principles to hardware documentation and implementation as well as mapping ITIL services to the underlying layers.

UPCOMING WEBINAR

We use cookies to ensure you get the best experience on our website.

LETS HANG!

FEATURED RESOURCES

TESTING
Mobile Software Testing eGuide
White Paper | StickyMinds

TESTING
The Care and Feeding of Your Software Testing Career
White Paper | StickyMinds

TESTING
Test Automation eGuide
White Paper | StickyMinds

TESTING
Internet of Things eGuide
White Paper | StickyMinds

Log In or Register to post comments

We use cookies to ensure you get the best experience on our website.

0 comments

YOU MAY ALSO LIKE

TESTING

Why Your AI Testing Suite Isn't What It Was Touted to Be

Nahla Davies | August 26, 2026

TESTING

Exploratory Testing Isn't a Junior Skill—You've Just Been Letting Juniors Do It

Nahla Davies | August 21, 2026

TESTING

Choosing the Right Testing Tool: How the Idea-t Framework Can Help

Isabel Evans | August 19, 2026

Join the crew for FREE! Access exclusive content, contribute, and more.

See All Member Benefits

We use cookies to ensure you get the best experience on our website.

STICKYMINDS
coveros

We use cookies to ensure you get the best
experience on our website.