Converting from ClearCase to McCabe CM - TRUEchange

1.0 Overview

This document will describe how to move the data from a ClearCase system to the TRUEchange system, given the two systems have very different underlying technologies.

The ClearCase SCM tool is an enterprise-wide software configuration management tool that was designed to handle large volumes of changes and files. Its branching capabilities are somewhat inefficient, but have been improved with addition of the UCM component. The UCM component allows the user to see the "streams" in a graphical way.

Some of the drawbacks of ClearCase are that it uses NFS-style connectivity, which does not allow for WAN configurations easily. Also, the servers (referred to as "VOBS") are very resource intensive, people are inclined to keep all of their data on directories in one server rather than broken up into logical projects on different servers.

The TRUEchange SCM tool is an enterprise-wide software configuration management tool that was designed to not only hold large amounts of change and many files, but it is also designed to properly handle changes as atomic units and branching as part of the fundamental building blocks of the product. No special additions are required to see the branches (referred to as "streams") and with the "integrated difference" technology.

2.0 ClearCase Elements as They Map to TRUEchange Elements

Before the conversion can begin, it is important to understand how the ClearCase data is defined within the database. We will relate this data to how it will map in the TRUEchange database after conversion.

The basic elements of the ClearCase system are files, file revisions, branches and labels. Files are loaded into an initial baseline, and work is done from there. Files are checked out, changed and checked back in, creating a revision for a file. Each change of a file is independent of each other. So changes to multiple files that are done for the same purpose; to implement a feature, to fix a bug, etc, are not tightly coupled, or atomic. For most setups, ClearCase branches are made to allow isolated work to be done on files without polluting the mainline. This work can later be merged back to the mainline when the work is completed. When a code is ready to be built and released, the file revisions that make up the work are applied a label.

The following is an example of a view of what a progressed development cycle would look like. This graph was made using ClearCase without UCM, so it has to be maintained manually. Looking at the revision graph of a single file in standard ClearCase does not necessarily give you the full product history:

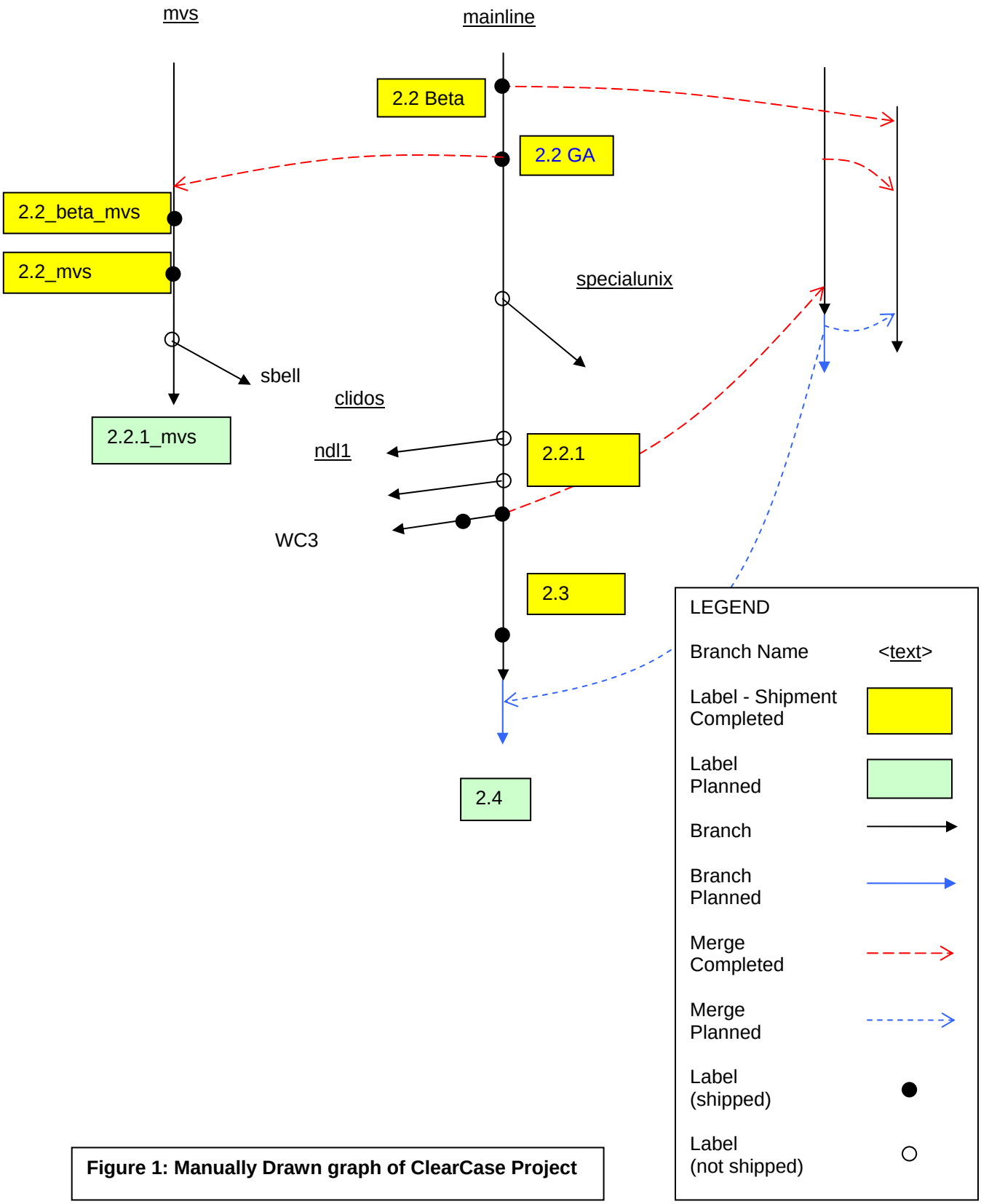
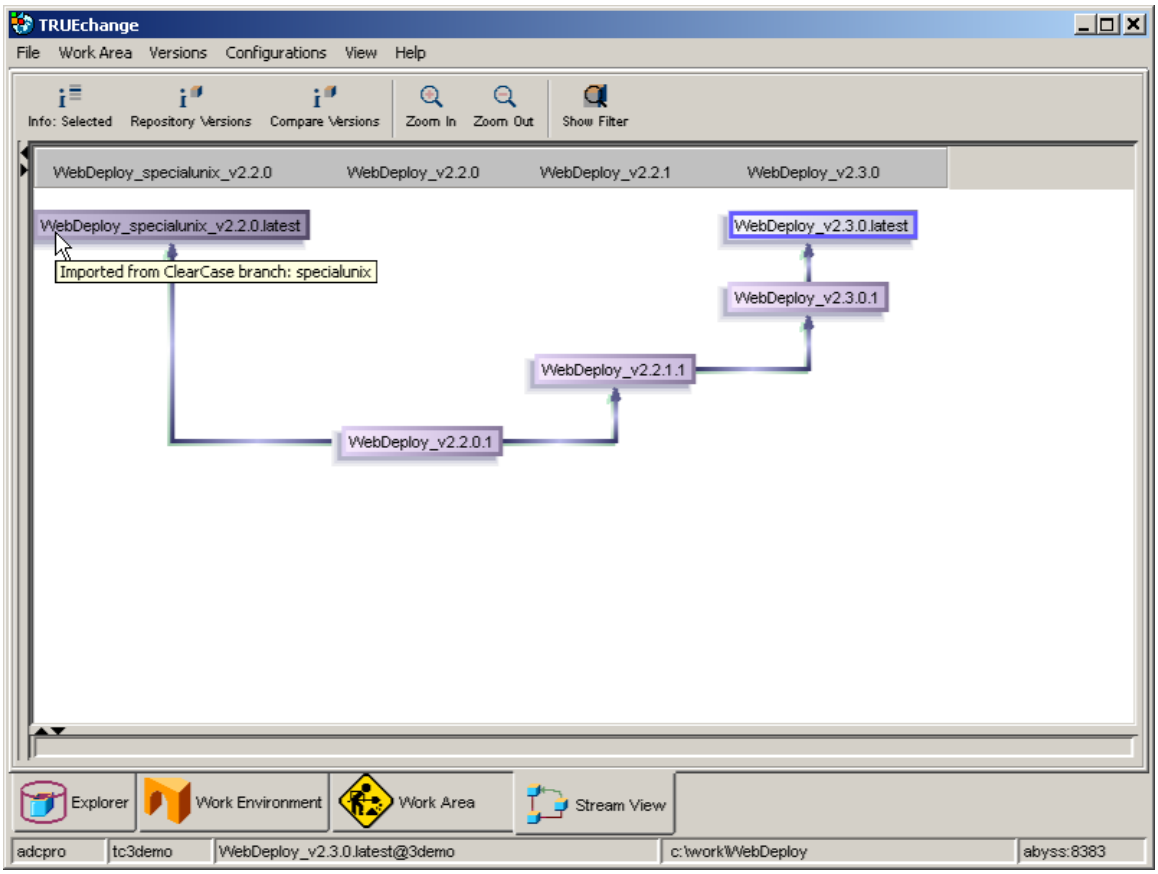


Figure 1: Manually Drawn graph of ClearCase Project

McCabe Software, Inc.
(800) 638-6316 ● URL- http://www.mccabe.com ● 2374 Post Road ● Warwick, RI 02886

In TRUEchange, the paradigm is slightly different. The elements that make up the system are files, integrated difference changesets, streams and checkpoints. Like ClearCase, the files are initially loaded into an initial version of the system. Files are also checked out and changed and checked back in, but the files do not have to be independently changed. The changes are encapsulated into integrated difference changesets. Unlike what some people have come to know as changesets, integrated difference changesets are structurally and architecturally different. We will discuss the importance of this distinction later. For the conversion process, we can think of them as just file revisions. Streams are similar to ClearCase branches. They get created when they are needed. They are much more structured than the free-form branches used in ClearCase. TRUEchange has checkpoints and streams instead of labels. Each stream can be frozen (checkpointed) at different points in time as changes are applied to them. Since creating streams is such a trivial task in TRUEchange, another thing to consider is making a stream where you may have just laid down a label in ClearCase. For example, in Figure 1, labels were applied on the mainline ClearCase branch for 2.2, 2.2.1, 2.3, etc. In TRUEchange, each release can have a separate development stream for the releases v2.2.0, v2.2.1, v2.3.0, etc.

Below is a screenshot of the StreamCM tool's Stream View:



2.1.1 Figure 2: Actual TRUEchange Graph From StreamCM Tool

So to recap, we now have the mapping of the ClearCase to TRUEchange elements so that we can make sense of an import of one system to the other:

TRUEchange	ClearCase
File	File
Integrated Difference Changeset	File Revision
Stream	Mainline/Branch/Label
Checkpoint	Label

Now we can move on to the conversion process itself.

3.0 Exporting From ClearCase and Importing to TRUEchange

When moving from a tool like ClearCase to TRUEchange, it is important to keep in mind the differences in architectures – deltas vs. integrated difference. For most delta tool to delta tool conversions, the process involves fewer steps because you are merely moving similar meta-data and file revisions into another mechanism that matches the previous tool's mechanism, thus more changes are easily captured, but you are not gaining any real benefit from moving tools. Perhaps one has a fancier interface, etc. But you are not gaining much other benefit.

The most efficient way to get the data from the ClearCase server to the TRUEchange repositories is to get just the changes from the major labels, whether they are on ClearCase branches or the mainline. Much of the intermediate delta file revisions can be ignored, because over time they will lose meaning anyway. The major labels were what were released, and they are what you will most likely ever need to reproduce again, or see why changes were made.

As you move forward with integrated difference changesets, the individual changes always can remain significant because:

1. They are closely bound to tasks
2. They can be **individually migrated between any stream**
3. They can be **removed from any stream**
4. The become self-documenting changes to the release

So again, using the example from Figure 1, if our project was called WebDeploy, the following process would be used if we wanted to start at capturing the history of the mainline from release 2.2 and forward. In this example, the file revisions from the ClearCase 2.2 label will become the baseline in the TRUEchange repository.

3.1 Exporting File History From ClearCase

From ClearCase:

1. Set the ClearCase configspec to look at the 2.2 Label
2. Copy out all the files to a snapshot view with a unique directory name such as WebDeploy_22
3. Set our ClearCase configspec to look at the 2.2.1 Label
4. Copy out all the files to a snapshot view with a unique directory name such as WebDeploy_221
5. Set our ClearCase configspec to look at the 2.3 Label

6. Copy out all the files to a snapshot view with a unique directory name such as WebDeploy_23
7. Set our ClearCase configspec to look at the tip of the mainline
8. Copy out all the files into a snapshot view with a unique directory name such as WebDeploy_mainline_tip

If you wish to convert any ClearCase branches into TRUEchange streams, repeat the process of setting the ClearCase configspec and copying out a snapshot view for each ClearCase branch that you wish to import into a TRUEchange stream. For example: Set the ClearCase configspec to look at the specialunix Label, and copy out all the files into a snapshot view with a unique directory name such as WebDeploy_specialunix.

3.2 Importing File History to TRUEchange

From TRUEchange:

1. Create a new TRUEchange project with initial project version: WebDeploy_v2.2.0.latest (where the phase letter “v” indicates a released stream)
2. Create a TRUEchange work environment for project version: WebDeploy_v2.2.0.latest and work directory: WebDeploy_22 (i.e. the directory containing the snapshot view for the ClearCase 2.2 Label)
3. Use Add Files to load all the files from the WebDeploy_22 directory into the TRUEchange WebDeploy_v2.2.0.latest stream
4. Checkpoint WebDeploy_v2.2.0.latest to create WebDeploy_v2.2.0.1 (you may wish to include the ClearCase 2.2 Label name in the checkpoint description)
5. Use Create Version to create a new stream named WebDeploy_v2.2.1.latest, and specify the parent version: WebDeploy_v2.2.0.1
6. Edit the TRUEchange work environment to point to the newly created stream: WebDeploy_v2.2.1.latest and the work directory: WebDeploy_221 (i.e. the directory containing the snapshot view for the ClearCase 2.2.1 Label)
7. Use Update Load to load all of the files from the WebDeploy_221 directory into the TRUEchange WebDeploy_v2.2.1.latest stream
8. Checkpoint WebDeploy_v2.2.1.latest to create WebDeploy_v2.2.1.1 (you may wish to include the ClearCase 2.2.1 Label name in the checkpoint description)
9. Use Create Version to create a new stream named WebDeploy_v2.3.0.latest, and specify the parent version: WebDeploy_v2.2.1.1
10. Edit the TRUEchange work environment to point to the newly created stream: WebDeploy_v2.3.0.latest and the work directory: WebDeploy_23 (i.e. the directory containing the snapshot view for the ClearCase 2.3 Label)
11. Use Update Load to load all of the files from the WebDeploy_23 directory into the TRUEchange WebDeploy_v2.3.0.latest stream
12. Checkpoint WebDeploy_v2.3.0.latest to create WebDeploy_v2.3.0.1 (you may wish to include the ClearCase 2.3 Label name in the checkpoint description)
13. Edit the TRUEchange work environment to point to the work directory: WebDeploy_mainline_tip (i.e. the directory containing the snapshot view for the tip of the ClearCase mainline). Leave the project version setting as is, it should still point to: WebDeploy_v2.3.0.latest

14. Use Update Load to load all of the files from the WebDeploy_mainline_tip directory into the TRUEchange WebDeploy_v2.3.0.latest stream
15. WebDeploy_v2.3.0.latest is now the active tip of the release 2.x development stream. Use Retire Versions to retire the other 2.x streams because these streams no longer need an active tip: WebDeploy_v2.2.0.latest, WebDeploy_v2.2.1.latest

If you wish to convert any ClearCase branches into TRUEchange streams, repeat the following steps for each ClearCase branch label that you exported from ClearCase:

1. Use Create Branch to create a new TRUEchange branch stream, where the TRUEchange branch name is similar to the ClearCase Label that will be imported to this TRUEchange branch stream, and the parent version is the checkpoint that corresponds to the ClearCase label prior to the ClearCase branch. For example: Use Create Branch to create a branch named WebDeploy_specialunix_v2.2.0.latest, and specify the parent version: WebDeploy_v2.2.0.1
2. Edit the TRUEchange work environment to point to the newly created branch stream and the corresponding work directory. For example: Edit the TRUEchange work environment to point to the newly created branch stream WebDeploy_specialunix_v2.2.0.latest and the work directory: WebDeploy_specialunix (i.e. the directory containing the snapshot view for the ClearCase specialunix Label)
3. Use Update Load to load all of the files from the ClearCase specialunix directory into the TRUEchange WebDeploy_specialunix_v2.2.0.latest stream. WebDeploy_specialunix_v2.2.0.latest is now the active tip of the specialunix development stream.

4.0 Moving Forward With TRUEchange

Now that you have moved your data into TRUEchange, it is not time for business as usual now. Since you are now working with a new technology, it is time for some new practices.

The biggest adjustment involves thinking of each change in terms of logical change. In delta tools, you are not encouraged to think of changes you are making to a Stream as a task, you often think of it as changing a bunch of files. Taking a little extra time to make sure you only change what your cset says it is going to change is good practice, and it saves much time later.

- A well-crafted cset can be easily migrated to other streams of work with little difficulty.
- A well-crafted cset can be removed easily if it becomes problematic.
- Well-crafted csets become self-documenting bullet items for release notes.

Falling into old habits of changing individual files for the same problem, slipping several extra fixes into a cset that was designed to only make one specific change, checking out a file to revert code back rather than removing problematic csets or poorly describing the fixes being made to save a few minutes all lead to hour or day delays down the road.

With discipline, TRUEchange will end up saving developers a lot of time when it comes to patching old releases and working with parallel development streams.

5.0 Delta VS. Integrated Differences

What is the benefit of Integrated Difference Changesets vs. Delta Changesets? Delta technology has been around for a long time. It captures change by storing the original contents of the file, then change sections or the deltas, between that original version and the change.

With file deltas, the file is reconstructed by taking the original content, then building up the revision requested by applying the deltas required to make that revision. The drawback to this approach is that each delta is dependent upon the previous to create the version of the file you want. This becomes a drawback when you want to only take a particular change in a file revision that was made in one branch of the code and apply it to another branch. The other drawback is that the branches themselves are defined by making copies of the original content and the changes to apply to it. So there is also inherent duplication in the "database".

With Integrated Difference technology, the file is comprised of a list of changesets that describe the file. The changeset itself is an entity that describes what lines are turned off or on in the file. The underlying architecture of how these lines are organized in the changeset are what make it independent of the changesets that chronologically come before or after it in any given stream. Since the streams are just a collection of changesets, and changeset are a collection of file line pointers, then there is no duplication of file content between streams. And it is their architecture that makes them able to be migrated independently between streams and removed from streams, regardless of the other changesets in the stream.

The classic description of the difference has been that deltas are like building blocks stacked up. You can't just pull one out to move it around. Whereas integrated difference changesets are like a deck of cards – where it is easy to add and remove cards without much impact on the deck.