Migration ClearCase® to Team Foundation Server

White Paper

Authors

Noel McKenna
Consultant
M. Sc. Computer Science and Applications
Fa. QbiQ AG
Noel.McKenna@qbiq.de

Christopher Rogall
Consultant
Computer Scientist
Fa. QbiQ AG
Christopher.Rogall@qbiq.de

Table of contents:

1 Introduction

1.1 Document Goal

The goal of this document is to create a white paper that is to provide guidance on:

- the migration of an existing Rational® ClearCase® repository or Versioned Object Base (VOB) to Microsoft's platform for Source Code and Version Control – Microsoft® Team Foundation Server® (TFS)
- the migration of an existing ClearCase work environment to a Microsoft TFS work environment
- the necessary organizational steps to be taken and their order of execution

These will combine to result in a smooth and efficient transfer of organizational productivity from the ClearCase environment to the TFS environment

1.2 Areas of Relevance

This white paper will cover the following migration topics:

- Comparison of ClearCase and TFS concepts
- Team Foundation Server Migration Tool for Rational ClearCase

The focus of this white paper is to outline a strategy for the migration of the data from ClearCase to a TFS.

1.3 Document Structure

The following contains an overview of the document structure:

- Chapter 2 includes a glossary of the terms and abbreviations that are used throughout this document.
- Chapter 3 will feature an analysis of the some ClearCase concepts and their TFS counterparts, and their relevance in a migration.
- Chapter 4 will focus on the Migration Tool for Rational ClearCase.
- Chapter 5 will feature an analysis and consideration of the Migration Tool functionality and contain an analysis of the issues as they relate to user management.
- Chapter 6 will focus on the User Account Management.
- Chapter 7 will present the Migration Roadmap.
- Chapter 8 includes the General Recommendations.
- Chapter 9 will show the linked appendix.

2 Glossary of Terms

The following is a list of terms and abbreviations that are used in this paper.

Term	Description
Rational® ClearCase®	Rational ClearCase – a solution that provides version control, workspace management, parallel development support and build auditing
Microsoft® Team Foundation Server® (TFS)	Microsoft Team Foundation Server or TFS - a Microsoft solution for source control, data collection, reporting and project tracking. **T**eam **F**oundation **S**erver is a Microsoft offering for source control, data collection, reporting, and project tracking, and is intended for collaborative software development projects. It is available either as stand-alone software or as the server side back end platform for Visual Studio Team System (VSTS).
VSTS	**V**isual **S**tudio **T**eam **S**ystem is a set of software development, collaboration, metrics, and reporting tools from Microsoft.
VOB	**V**ersioned **O**bject **B**ase is a repository that stores versions of file elements, directory elements, derived objects, and metadata associated with these objects. With MultiSite, a VOB can have multiple replicas, at different sites.
Branch	**Branches** organize the different versions of files, directories and objects that are placed under version control. A **branch** is an object that specifies a linear sequence of versions of an element
Label	A **label** is a clearly marked a name brand, known as the target.
Merge	**Merge/Merging** is the act of reconciling multiple changes made to different copies of the same file. Most often, it is necessary when a file is modified by two people on two different computers at the same time. Later, these changes are merged, resulting in a single new file that contains both sets of changes.
GUI	**G**raphical **U**ser **I**nterface is a type of user interface which allows people to interact with a computer and computer-controlled devices.
XML	E**X**tensible **M**arkup **L**anguage is a general-purpose specification for creating custom markup languages.
MVFS	**M**ulti**V**ersion **F**ile **S**ystem is a virtual file system which displays specific versions of data stored in ClearCase. In particular, it supports *dynamic views* which can show an arbitrary combination of local and remote files.

3 ClearCase and TFS Compared

ClearCase and TFS are version control systems that enable users to manage critical parts of the Software Development Cycle (SDLC):

- Version Control
- Work Item Tracking
- Reporting

Both systems function as repositories storing elements and the associated history data.

3.1 Branching

3.1.1 Branching Concepts

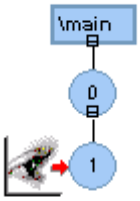
TFS and ClearCase use very different branching models.

In ClearCase branch instances are created just-in-time. Just-in-time branching means that an element will be branched only when it is checked out to be modified in an appropriate view by means of the “-mkbranch” rule in the config spec.

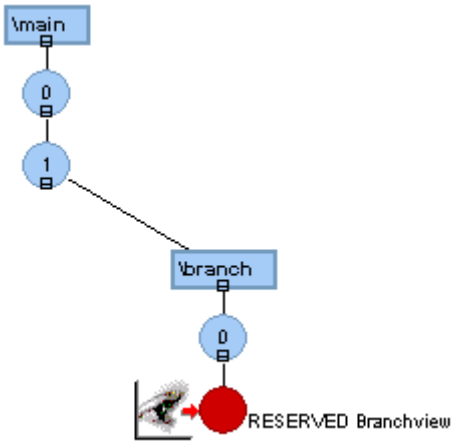
Elements that are not checked in such a view will not be branched

The branch in ClearCase will only contain the elements that have been checked out onto it.

File before check out onto branch



File checked out onto branch



Team Foundation Server branches in "path space", which is analogous to copying a folder in Windows Explorer. However, unlike copying, branching maintains the history relating the source to the target to allow merging future changes.

In TFS, a branch (directory) contains the entire contents of the source. All of the files/folders located in the source are branched at the time of branch creation.

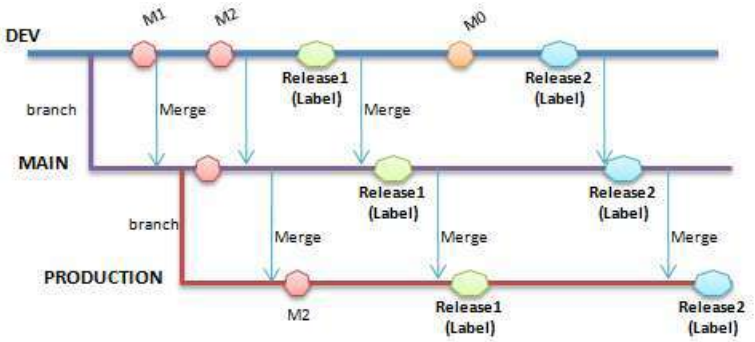
Branches in ClearCase are represented as folders in TFS. That is to say, what a view and a config spec make visible in ClearCase, is made visible in a folder in TFS. The versions of the elements that have a branch applied to them in ClearCase are migrated to a folder in TFS that possesses the same name as the ClearCase branch.

3.1.2 Which Branches should be migrated to TFS?

In some cases, ClearCase VOBs can contain a very large number of branches. This is often because many users will create many short lived branches used to fix bugs or add new features.

In TFS, the branching guidance is to keep a smaller number of branches, typically, MAIN, DEV, PRODUCTION, and the feature branches off of MAIN[1]:

- DEV – activity around new feature development and bug fixing
- MAIN (also called TEST or INTEGRATION) – activity around ensuring product stabilization and readiness for release
- PRODUCTION – activity around providing sustained engineering services for released products



In order to keep the time to migrate files at a reasonable level, and to avoid a complex branching structure in TFS, the guidance for migrating branches is to migrate those ClearCase branches that are synonymous to MAIN, DEV, and PRODUCTION. It may also be desirable to migrate some specific feature, release, or bugfix branches, but the rule of thumb should be to minimize the number of branches migrated to TFS.

3.2 Labels

The labelling concepts in ClearCase and TFS are similar.

3.2.1 Labeling in ClearCase

Labels are used to annotate element versions and to mark important configurations. Labels are usually applied to set of elements to mark an important project milestone or the starting point of a branch[2].

3.2.2 Labeling in TFS

Labeling in TFS results in the saving of a snapshot of the source coded based on either:

- Workspace version
- Timestamp

Each label is uniquely stored in TFS, enabling users to easily retrieve this source code snapshot. Example of such a snapshot scenario would be for a successful software build.

---

[1] Team Foundation Server Branching Guidance – John Jacob, Mario Ridriguez and Graham Barry Microsoft Corporation

[2] Redbooks: Software Configuration Management A Clear Case for IBM Rational ClearCase and ClearQuest UCM – IBM.com/redbooks

Depending on the permissions granted to specific users, labels can be:

- modified – files can be changed, added and removed from the label[3]
- moved – a label may be moved from one version of a file or folder to another version of the file or folder

The overuse of labels in TFS is with regard to the following observations not advisable:

- Team Foundation Server does not retain a history of changes made to the label.
- Given certain permissions, labels might be deleted or otherwise invalidated by changes, and there is no way of auditing those changes.
- There can be contention for a given label if more than one person wants to use and modify the label or the files contained in the label

3.2.3 Which Labels should be migrated to TFS?

It is recommended to use labeling in TFS only in cases where a short term snapshot of the source code is required. Long term use of labels in TFS is only recommended when permissions are strictly enforced. This strict enforcement ensures that changes to the labels are avoided, and even if the changes are necessary, they are at least controlled. During the migration it is possible to select all or a subset of existing labels for migration. The continued usage of these labels must be controlled in TFS. For development activities that use a snapshot of source code over a longer time period, it is recommended to use Branching in TFS.

3.3 Merging

The merge strategies in TFS and ClearCase are similar.

Merging in version control is the process of combining changes that have transpired in two distinct branches. A merge operation takes changes that have occurred in the source branch and integrates them into the target branch. Merging integrates all types of changes in the source branch including name changes, file edits, file additions, file deletions, and un-deletions.

3.3.1 Merging in ClearCase

A merge combines the contents of at least two or more versions of the same file or of the same directory into a single version of a new file or directory. The ClearCase merge algorithm uses the following files during a merge[4]:

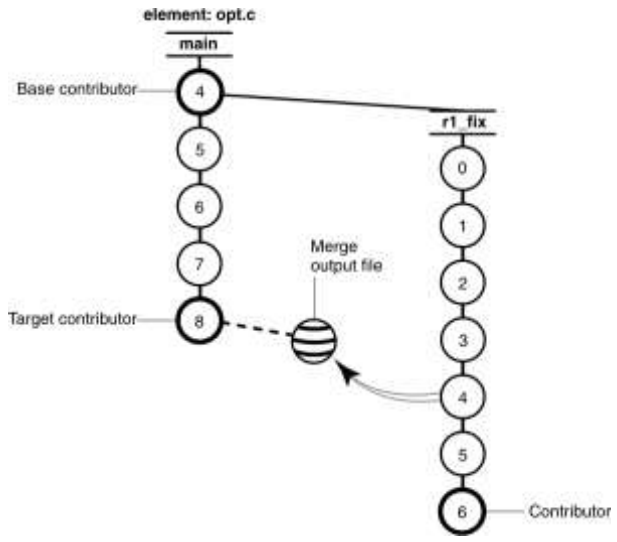
- **Contributors**, which are typically one version from each branch you are merging. (It is possible to merge up to 15 contributors.) The user specifies which versions are to be merged.
- The **base contributor**, which is typically the closest common ancestor of the contributors. (For selective merges, subtractive merges, and merges in an environment with complex branch structures, the base contributor may not be the closest common ancestor.) The merge algorithm determines which contributor is the base contributor.
- The **target contributor**, which is typically the latest version on the branch that will contain the results of the merge. The user determines which version is the target contributor.
- The merge output file or directory, which contains the results of the merge and is usually checked in as a successor to the target contributor. By default, the merge output file is the

[3] Team Foundation Server Branching Guidance – John Jacob, Mario Ridriguez and Graham Barry Microsoft Corporation

[4] Software development with base ClearCase - http://publib.boulder.ibm.com/infocenter/cchelp/v7r0m0/index.jsp?topic=/com.ibm.rational.clearcase.hlp.doc/cc_main/about_merge_base_cc.htm

checked-out version of the target contributor, but it is possible to choose a different file to hold the merge output.

- In the diagram below, the contributor branch is r1_fix, version 4 of this branch is selected as the contributor version



3.3.2 Merging in TFS

Merging is the logical corollary to branching in Team Foundation Server[5]. The merge engine is able to do its work because Team Foundation Server keeps a record of all changes including the relationships of a child branch to the parent branch.
If items are modified in both the source and target branches, a version conflict will be filed and the user is prompted to resolve those conflicts. In this case the conflict resolution dialog will launch a graphical three-way merge tool that assists the user in completing the merging activities.
Once a branch has been created, moving changes from the source branch to the target branch or from the target branch back to the source branch is straightforward. Team Foundation Server maintains the relationships of branched items and the merge history.
When merging across branches, all of the changes that have been made in the source and target branches are merged. This includes additions, deletions, un-deletions, and moves (which is effectively a rename operation). For example, if an item in the source branch has been renamed from A.TXT to B.TXT, when performing a merge that change will also rename the corresponding item in the target branch.
The merge becomes part of a set of pending changes that are contained within the users individual workspace. The entire set of merged changes is committed atomically as a single changeset. For more information regarding the topic changeset and its role in TFS see Appendix 2.

[5] Team Foundation Server Branching Guidance – John Jacob, Mario Ridriguez and Graham Barry Microsoft Corporation

4 TFS Migration Tool for Rational ClearCase

The TFS Migration Tool for Rational ClearCase migrates existing repositories in ClearCase to TFS[6]. The tool supports a total of three migration approaches:

1. Approach I – Snapshot Based Migration
2. Approach II – Selected Branch Migration without Branch Relationship
3. Approach III – Selected Label and Branch Migration with Branch Relationships

4.1 Migration Approach I (Snapshot based Migration)

Only a single visible version of the files / folders selected for the migration is transferred to TFS. Other versions of the elements and the metadata (labels, branches etc.) associated with elements **are not** migrated.

A config spec is provided to the tool prior to the execution of the migration. The elements that are made visible in ClearCase using this config spec are then migrated to TFS.

4.2 Migration Approach II (Individual Branch Migration)

All versions of the selected files / folders selected for the migration are transferred to TFS. The branches that are selected branches are also migrated to TFS. Labels are not migrated to TFS. No information regarding the relationship between files / folders belonging to the various branches is migrated.

The user selects the files/folders to be migrated along with the branches that are to be migrated. For each selected branch a config spec is provided to the tool prior to the execution of the migration, these config spec(s) are used by the tool to identify and migrate the versions of the elements that do have the selected branch applied to them– Missing File Processing. See Chapter 4 for more information.

4.3 Migration Approach III (Complete Migration with Branching)

Using this approach, the complete migration of the selected files/folders takes place along with all the metadata along the branching structure of the selected branches. Selected labels are also migrated. All the versions and all the version history is migrated to TFS.

See section 3.2 Migration Approach II (Individual Branch Migration) for a brief overview on branch migration.

Migration Approach III is the focus of this Whitepaper.

[6] TFS Migration Too for Rational ClearCase User Guide

4.4 Migration Tool Overview

The tool can be run using the Migration Tool User Interface:

4.4.1 Migration Tool Configuration using the GUI

Configuring the Migration Tool prior to running the migration of sources from a ClearCase VOB to TFS:

On selecting the Settings option, the user may configure the Migration Tool:

The ClearCase Server settings to be configured are:

Setting	Description
View Type	Only Dynamic view is created by the tool for the purposes of migration.
MVFS Drive	The drive to use to connect to the view being created,
ClearCase View Storage Directory	The ClearCase view is created here during the migration
Temporary Workspace Folder	The location where the Migration Tool will download the files from TFS during syncing between TFS and ClearCase

The Team Foundation Server settings to be configured are:

Setting	Description
Temporary Workspace Folder	The location where the Migration Tool will create the Team Foundation Workspace during the migration. This location will be used for downloading files from ClearCase during the migration

The Migration settings to be configured are:

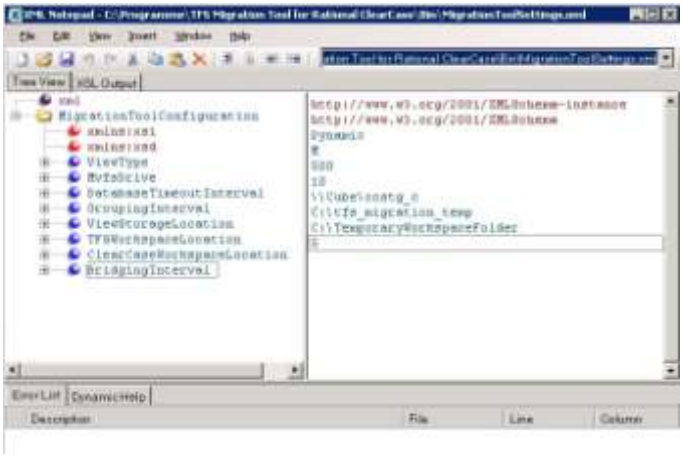
Setting	Description
Database Server Name	The database server that will be used for the migration.
Authentication	The authentication type for connecting to the database server – only Windows Authentication is supported.
Database Timeout Interval	The time for which the database server will be polled when trying to establish a connection.

The Bridging settings to be configured are:

Setting	Description
Bridging Interval	The time interval in which the migration will be bridged.

4.4.2 Migration Tool Configuration using XML

The configuration of the Migration Tool need not be completed using the GUI. All the settings that are configurable using the GUI in Migration Tool Configuration using the GUI are also configurable using the XML file MigrationToolSettings.xml:



4.4.3 Step-by-Step guide to Migrating from ClearCase to TFS using the GUI

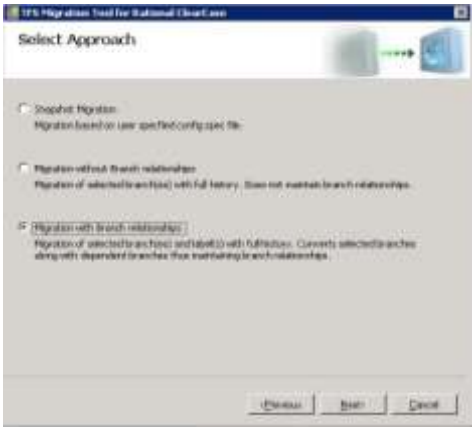
ClearCase Requirements:

- A valid MVFS (Multi Version File System) drive letter
- A valid ClearCase Storage Directory
- A VOB that contains version controlled sources
- ClearCase Version 2003.06.10 (doesn´t work with V7)

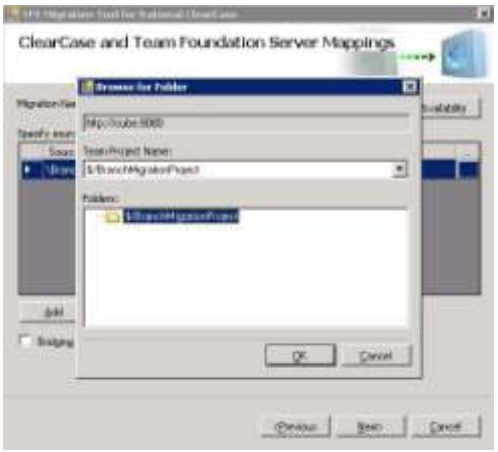
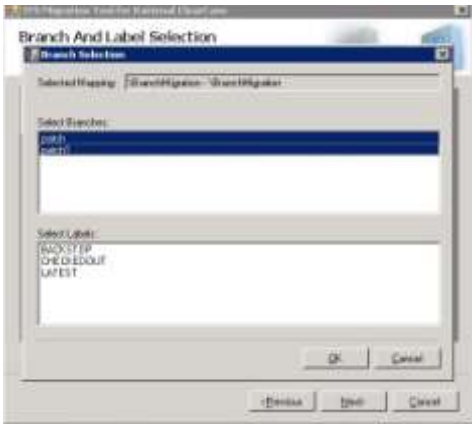
TFS Requirements:

- A valid Team Foundation Server Instance (TFS 2005)
- TFS Client 2005
- SQL Server 2005
- A TFS Team Project

Executing Migration Approach III (Complete Migration with Branching)[7]:

Migration Action / Step	Description
1. Select the option 'Start New Migration'	--
2. Select one of the three Migration Approaches (choose Approach III Complete Migration with Branching):	

[7] For a comprehensive guide to executing a migration and more information regarding the ClearCase Migration Tool and its features please see appendix 1

Migration Action / Step	Description
3. Configure the ClearCase Server	Enter the name of the ClearCase server and select 'Windows Authentication'.
4. Enter a valid Team Foundation Server URL	Enter the URL of the TFS Server and select 'Windows Authentication'.
5. Enter a valid name for the migration and select the ClearCase VOB(s) and Folder(s) that are to be migrated to TFS	The entered name will be validated for its authenticity by the Migration Tool It is possible to enter multiple ClearCase VOBs and folders.
6. Map the selected VOB(s) and folder(s) to team project(s) in TFS:	
7. For each mapping select the labels and branches that are to be migrated:	
8. For each selected branch, select a valid config spec. It is also possible to select one config spec which is valid for all mappings. The config specs enable the Migration Tool to identify and migrate those files that do not have the branches selected for migration.	

Migration Action / Step	Description
	
9. Specify the file path, were the approach XML file as per the selected/specified options until now will be saved.	On specification of the path file location and selection of the 'Next' button, the user confirms the migration settings. The migration tool input file will be generated.
10. Start the pre-processing pass.	Issues may be encountered during this pass, the user may choose a default resolution by the tool or the issues may be resolved by the user using the Reporting Module. If the Migration Tool finds any branches other than the branches that have been selected by the user in Step 7, that have to be migrated in order to maintain the branching hierarchy, then the user will be asked to enter the relevant config specs at the end of this pass.
11. On completion of the pre-processing pass (either automatically or after some user actions in the case of issues arising), the Migration pass starts automatically:	--
12. The Migration is complete and the sources are present in TFS:	

4.4.4 Migration Tool XML Output

4.4.5 Approach XML

The Approach XML file stores the migration settings as entered by the user in the GUI when running a migration. The Approach XML file may be referenced by the user when starting future migrations, thus ensuring that the user does not have to repeatedly enter information in any future migrations.

Please see Appendix 1 for more information regarding the approach XML generation.

4.4.6 Report XML

The Migration Tool generates an XML file for each migration.

This file is contains statistics regarding the migration.

Please see Appendix 1 for more information regarding the report generation feature of the Migration Tool.

4.4.7 Migration Tool Reporting Website

The Migration Tool features a reporting website. This website contains information on all previously run migrations

The issues, errors, warnings and information that are related to each migration can be examined using the reporting website. Indeed Issues that may have emerged during the pre-processing phase of migration Approach I and II may be resolved using this reporting website. See Chapter 5.2 Multiple Contenders Technical Example for more information.

Please see Appendix 1 for more information for more information regarding the Migration Tool and its reporting website functionality.

5 Migration Tool Functionality during the Migration

5.1 Migrating Branches and Missing File Processing

5.1.1 Overview

There is a basic difference between how branching is performed in ClearCase as compared to TFS – see Chapter 2 ClearCase and TFS Compared. In order to ensure that the same numbers of files are visible in a branch in both ClearCase and TFS, the concept of Missing File Processing has been introduced. The config spec that is given for each branch in ClearCase is used to enable the Migration Tool to migrate those files that are 'missing' for that branch. This ensures for example that builds that ran successfully on a ClearCase view using this same config spec also will also run successfully on its equivalent branch in TFS.

5.1.2 Missing File Processing Overview

The disparity between ClearCase and TFS branching leads to some challenges when migrating branches. Configuration specifications are integral to determining the correct files to select for a branch in ClearCase[8]. For this reason, the ClearCase migration tool requires a config spec for each branch that is to be migrated. Using this config spec, the tool is able to create TFS style branches out of the ClearCase branches. This, in effect, means that files that were never been branched now exist on the TFS folder that is created for the branch by the Migration Tool.

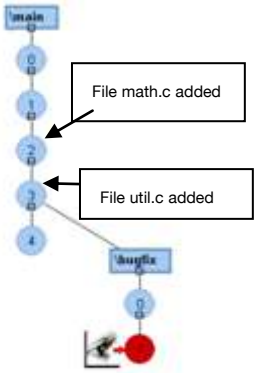
Although this strategy is not truly preserving the data as it existed in the ClearCase branch, it is a necessary transformation to ensure the completeness of branches in TFS. For example, without all of the correct (missing) files on the branch in TFS, the files will no longer successfully build. Merging files back into parent branches is also impossible without all of the correct files in TFS, essentially defeating the reason to branch in the first place. For these reasons, the tool takes the liberty of adding the missing files to the branch, as specified in the config spec.

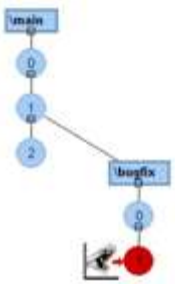
5.1.3 Technical Example

5.1.4 Migration Scenario

Using the following config spec, the sources in ClearCase look like this:

Config spec **Folder Source** **File math.c** **File util.c**





[8] Migrating ClearCase Branches to TFS http://blogs.msdn.com/tfs_migration/ Matt Mitrik

The following files can be seen through the view as specified by the config spec:

Source (/main/bugfix/1)

+----- math.c (/main/bugfix1)

+----- util.c (/main/4)

5.1.5 Expected Migration Results

The file util.c does not have a bugfix branch applied to it, this is due to the fact that the element was never checked out and checked in using the view as specified by the config spec. If the Migration Tool did not have the config spec as input information, it would not be able to migrate the util.c file to TFS, when using Approach II or Approach III. Following this logic the contents of the bugfix folder in TFS ***could be***:

Source (/main/bugfix/1)

+----- math.c (/main/bugfix1)

This ***would*** result in the contents of the bugfix branches differing in ClearCase and TFS.

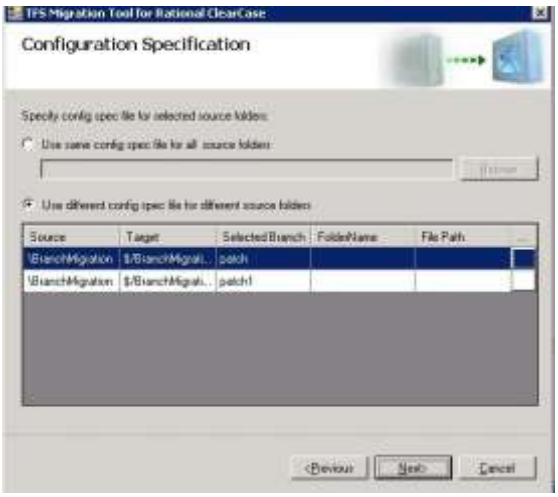
For this reason *Missing File Processing Rules* were introduced.

All files that satisfy each of the following two conditions for a branch (branch X) are considered missing for this branch in TFS:

1. The file does not have branch X applied to it
2. The file has not been added to the branch X of the parent folder[9]

In the above example, util.c does not have the branch bugfix applied to it AND it has not been added to the bugfix branch of the parent folder 'Source'. Util.c was added in the main branch of folder 'Source'. Therefore the Missing File Processing rules come into play during the migration.

Missing File Processing enables the Migration Tool to process the config spec that is passed to the Migration Tool during the execution of the migration:



[9] If either of these two conditions are true, the file will be migrated without the need for the Missing File Processing Rules coming into play. For example, if a file has been added to the bugfix branch of (parent) folder source and if this file does not have a bugfix branch applied to it, the Migration Tool is able to migrate it to TFS without the Missing File Processing Rules coming into play. This due to the fact, that it was added to the bugfix branch of the parent folder Source.

The file versions of all elements made visible in ClearCase by this same config spec are copied to the TFS bugfix branch folder. This result's in main/LATEST version of file util.c being copied to the bugfix folder in TFS, thereby ensuring that the all elements that were visible on the bugfix branch/config spec in ClearCase, are also present in the TFS bugfix branch. This is made possible by selecting the config spec as used in ClearCase for the bugfix branch and passing it to the Migration Tool during executing of the migration.

This config spec enables the Migration Tool to locate the files that do not have branch bugfix applied to them, but were visible in ClearCase and add these files in the bugfix folder in TFS as a missing version.

5.1.6 Migration Results Example Conclusion

The util.c file in the Source folder was *missing* for the branch bugfix (util.c did not have have the bugfix branch applied to it *AND* util.c had not been added to the bugfix branch of the parent folder Source). The Migration Tool is able to use the Missing File Processing Rules together with the config spec and branch the elements in TFS that have not already been branched just-in-time in ClearCase, e.g. util.c.

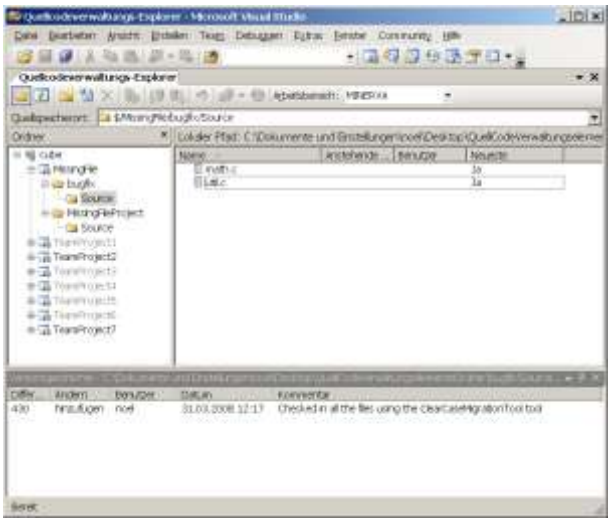
The contents that were visible in ClearCase using the config spec for the bugfix view are present in the TFS equivalent to this bugfix branch, namely the bugfix folder in TFS:

Util.c represents such a 'missing file' to the Migration Tool, thus the Migration Tool uses the Missing File Processing Rules and the config spec and adds the util.c file version (main/LATEST) to the "bugfix/source" folder in TFS:

Source (/main/bugfix/1)

+----- math.c (/main/bugfix1)

+----- util.c (/main/4)



The migration tool branches all the folders from their 0th version. This ensures that when a folder version is branched, the folder is empty, **thus avoiding** the following potential error conditions:

- When the Migration Tool creates a branch 'bugfix' from the Source folder /main/3 version math.c and util.c in TFS, both files will be copied to the bugfix folder in TFS. Due to this copy (and if the migration tool **did not branch** all folders from their 0th version), if an attempt is made to branch another version of math.c (for example math.c version /main/4) to the bugfix folder an error message saying that the file is already present would occur.
- The same logic would apply if an attempt was made to branch another version (e.g. util.c version /main/3) of the util.c file.

5.2 Migrating Labels

5.2.1 Overview

The Migration Tool will successfully migrate the selected labels and store them in TFS.

Using the label to access the snapshot of the sources should result in the same sources being made visible to the user in TFS as were made visible to the user in ClearCase.

- Labels on file elements are correctly migrated.
- Labels on directory elements are not migrated (please see Technical Example below for further information regarding the migration.

5.2.2 Technical Example

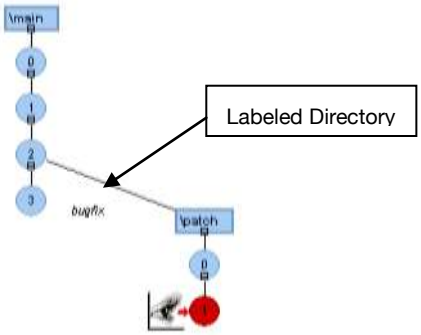
5.2.3 Migration Scenario

There is a labeled directory. This directory comprises a folder ‘SourceCode’. The goal is to migrate this labeled directory to TFS. The same information as was visible to the user in ClearCase must be visible to the user in TFS.

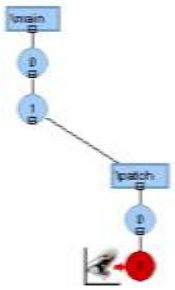
Graphical Version Trees of the elements (files) to be migrated:

Directory structure in ClearCase:

Folder ‘SourceCode’ with label:



File ‘Code.pl’ in folder ‘SourceCode’ without label:



5.2.4 Expected Migration Results

The migration of several branches with several labels attached to several different files should result in these labels being present in TFS as they were in ClearCase:

- The following directory structure is expected to be created:

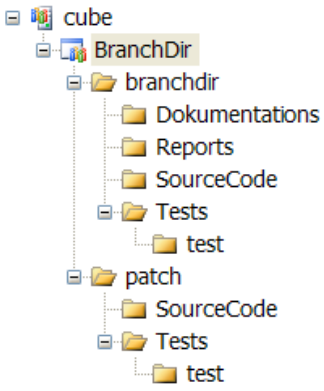
  Branchdir (=main)
  +----- Dokumentations
  +----- Reports
  +----- SourceCode
  +----- Tests
  +----- test

  Patch (=branch)
  +----- SourceCode
  +----- Tests
  +----- test

- The folder „SourceCode“ should be present as follows in TFS:
  - The ClearCase version „\main\3“ should not possess the label „bugfix“ in TFS ‘Branchdir/SourceCode’

5.2.5 Migration Results

Directory structure in TFS:



- The label ‘bugfix’ as was present in ClearCase for directory “SourceCode” version “\main\3” has not been migrated by the Migration Tool. Labels on folders are not migrated. This is due to the Changeset creation that the Migration Tool executes when migrating sources from ClearCase to TFS. See Chapter 6 Migration Tool Functionality regarding TFS Features for more information on this topic.

5.3 Migrating from ClearCase and Branching Ambiguity Issue

Branching Ambiguity comes into play during the migration when a file version tree consists of an empty /main/0 version and a further branch version or versions. This occurs when an element is created in a view and due to the config spec of this view, a versioned branch containing no /main/1 version is created. This occurs in ClearCase when the Multiple-Level Auto-Branching concept is used for making branches.

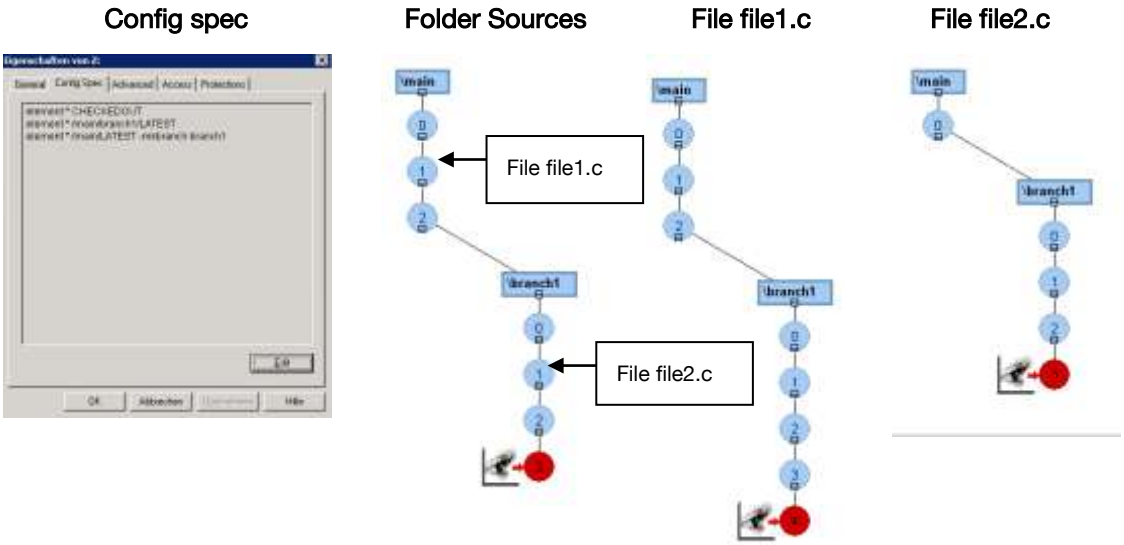
Due to the empty /main/0 version, the file (file2.c) in the example below consists of a branch (branch1) that also serves as its target folder (the folder in which the main versions of the file will be

mapped to) in TFS. See Chapter ‘Branching in TFS’ for more information regarding the branching concept in TFS.

Normally a file does not have an empty /main/0 version, in this case the target folder (the folder in TFS in which the branch versions will be mapped to) is different to the folder in which the main versions of the file will be mapped to. E.g. file1.c below, the main versions of this file will be mapped to folder ‘Sources’ in TFS and the branch1 versions of the file will be mapped to folder ‘branch1’ in TFS.

5.3.1 Technical Example

ClearCase Scenario:



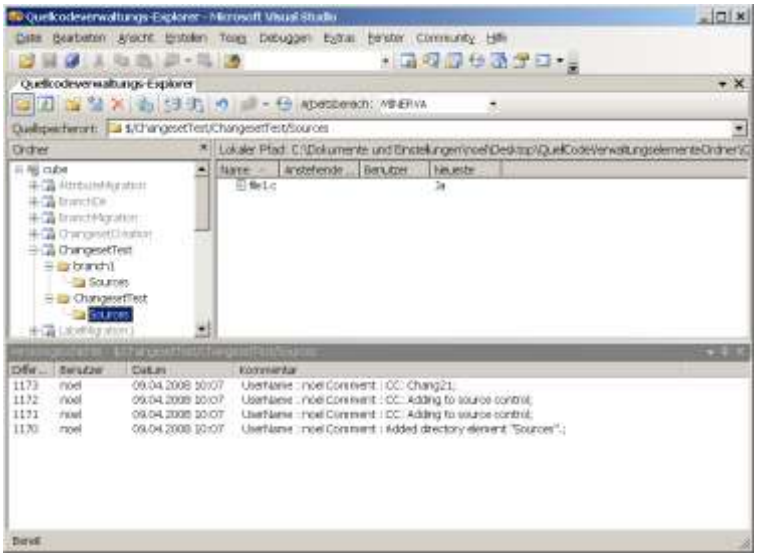
5.3.2 Expected Migration Results

1. File1.c
    - Analysis of File1.c:
        - File1.c was added in the main branch of folder Sources
        - Target folder in TFS (the folder in TFS in which versions of the main branch of the file will be mapped to) is ‘Sources’.
        - File1.c contains a branch ‘branch1’, the versions of file1 on this branch are expected to be mapped to folder in TFS called ‘branch1’.
    - Analysis Conclusion:
        - The file File1.c does not contain a branch that also serves as its target folder in TFS. It’s main branch contains versions and is not empty, thus the Migration Tool will not encounter Branching Ambiguity Issues and the branch ‘branch1’ will not be ignored during the migration.
2. File2.c
    - Analysis of File2.c:
        - File2.c was added in branch branch1 of folder Sources.
        - Target folder in TFS (the folder in TFS in which the main branch of the file will be mapped to) is ‘branch1’.
        - File2.c consists of an empty main branch and another branch ‘branch1’, the versions of file2 on this branch are expected to be mapped to folder in TFS called ‘branch1’.

- Analysis Conclusion:
  - The file file2.c contains a branch ‘branch1’ that also serves as its Target folder in TFS – ‘branch1’. This is due to that fact that the main branch of the file is empty, this results in the Migration Tool encountering a Branching Ambiguity Issue during the migration of file file2.c. This in turn means that the versions of the file File2.c on branch ‘branch1’ will be ignored during the migration.

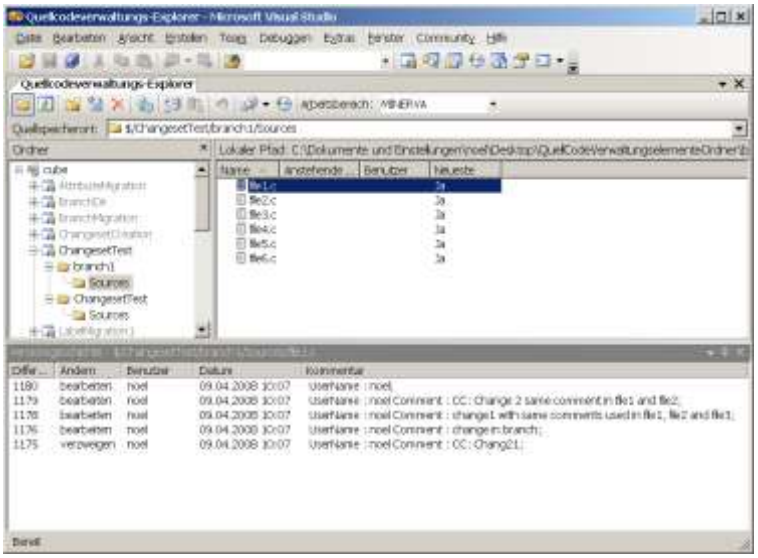
5.3.3 Migration Results and Branching Ambiguity

File1.c is migrated to TFS (main versions mapped to folder Sources):

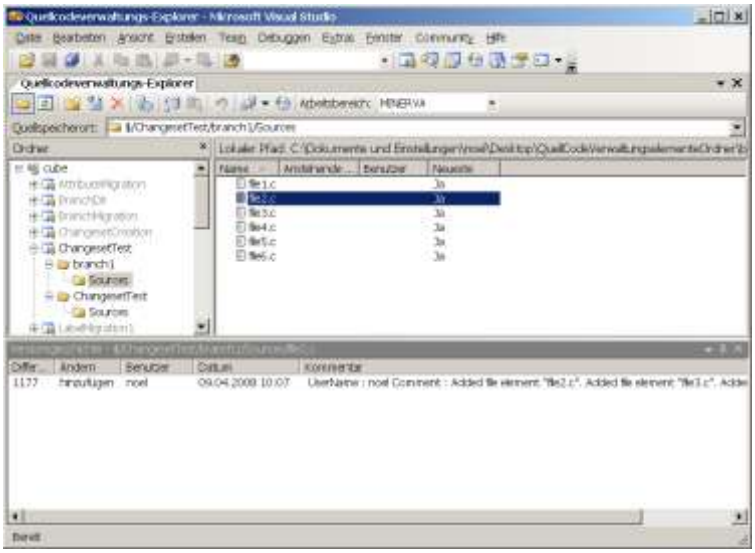


File1.c is migrated to TFS:

(branch1 versions mapped to folder branch1):

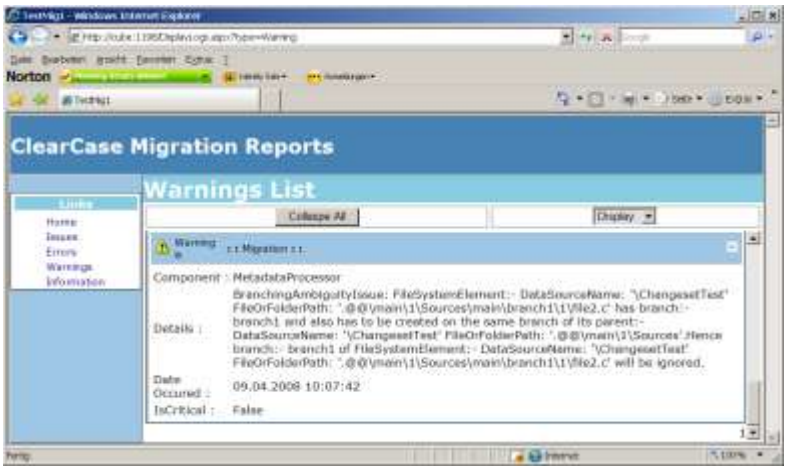


Versions of File2.c on branch1 are ignored during the migration to TFS (branch1). Only the ClearCase /main/0 empty version of the file is migrated to folder branch1:



5.3.4 Migration Results Example Conclusion

The encountered Branching Ambiguity Issue when migrating file2.c is reported on the Migration Tool website. The migration of the branch1 versions of file2.c may be carried out manually at a later stage:



5.4 Migrating from ClearCase and Multiple Contenders Issue

The issue of multiple contenders arises when **different** ClearCase elements **sharing the same name** and **having the same branch type applied** to them are to be migrated to TFS.

This scenario can arise when:

- An element named ‘file1.c’ is added in the main branch of a folder ‘folder1’. The element ‘file1.c’ has a branch named ‘bugfix’ applied to it.
- ***Another*** element, also named ‘file1.c’ is added in the ‘bugfix’ branch of ‘folder1’. This element ‘file1.c’ also has a branch named ‘bugfix’ applied to it.

The two elements ‘file1.c’ are to be migrated to TFS. The bugfix branches of the two elements sharing the same name will be mapped to a folder named ‘bugfix’ in TFS. See Chapter 2 ClearCase and TFS Compared for more information on branching. This results in two identically named elements in ClearCase, having the same branch type applied to them competing for the same folder in TFS – Multiple Contenders Issue.

5.4.1 Technical Example

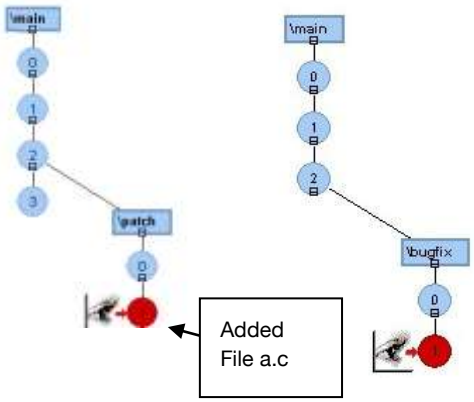
This example is referred to in ClearCase circles as being the ‘evil twins’ scenario. The occurrence of this scenario using ClearCase is normally prevented by use of a mkelem- trigger, which prevents the creation of two elements sharing the same name.

ClearCase Scenario:

1. A folder ‘Foo’ has two branches:
    a. Main
    b. Patch
2. a.c file element 2 added in ‘patch’ view:

a.c **Config spec** **Folder Foo** **File a.c**

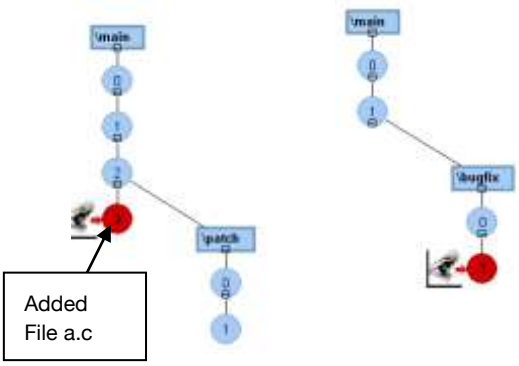
1. Has been added in branch ‘patch’ of Foo
2. Has branches ‘main’ and ‘bugfix'
3. Shares the same name as the element added in the example above.



3. a.c file element 1 added in ‘bugfix’ view:

a.c **Config spec** **Folder Foo** **File a.c**

1. Has been added in branch ‘main’ of Foo
2. Has branches ‘main’ and ‘bugfix'
3. Shares the same name as the element added in the example below.



5.4.2 Expected Migration Results

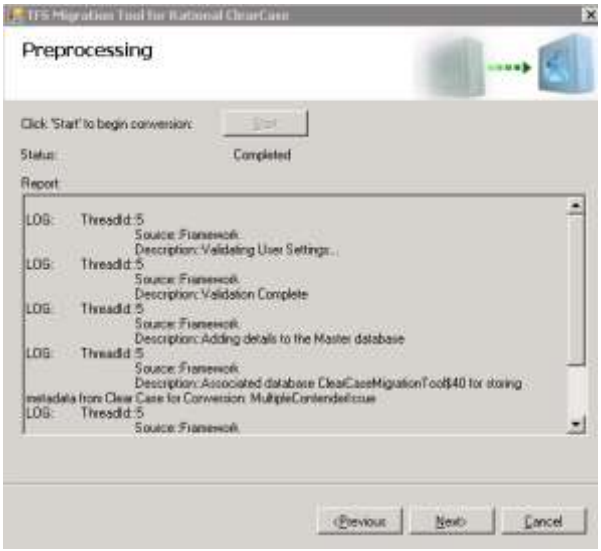
The Migration Tool creates a folder named ‘bugfix’, the versions of the elements in ClearCase that have the branch bugfix applied to them will be migrated to this folder during the migration. The two identically names a.c files are two such elements that have the branch bugfix applied to them in ClearCase:

- a.c added in branch ‘main’ of folder ‘Foo’
- a.c added in branch ‘patch’ of folder ‘Foo’

There are two elements that share the same name and these same two elements are to be migrated to the folder ‘bugfix’ in TFS. The Migration Tool requires input from the user in order to decide which a.c file element bugfix branch to migrate.

5.4.3 Migration Results and Multiple Contenders

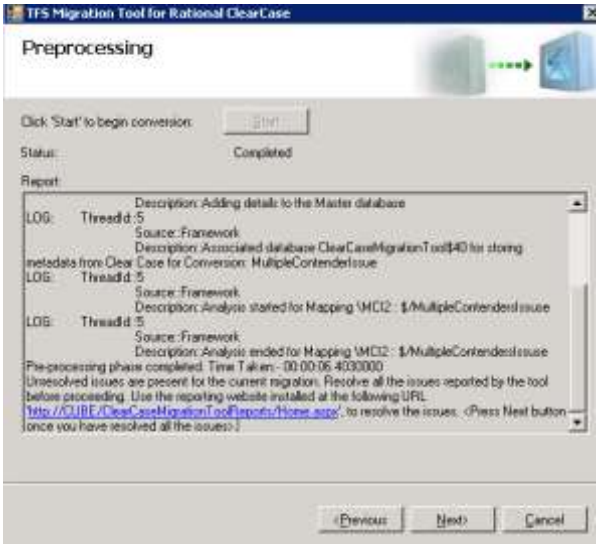
1 During the preprocessing phase of the migration:



2 The migration tool informs the user that there are multiple contenders issues present in the current migration:

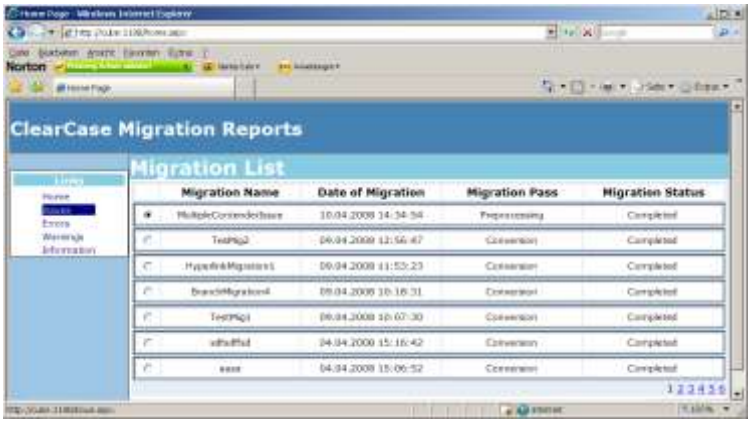
3 The user can choose to resolve the issues (Select Ja button) or allow the migration tool to generate a default solution to the multiple contenders (Select Ja button).

4 Select the 'Ja' button:

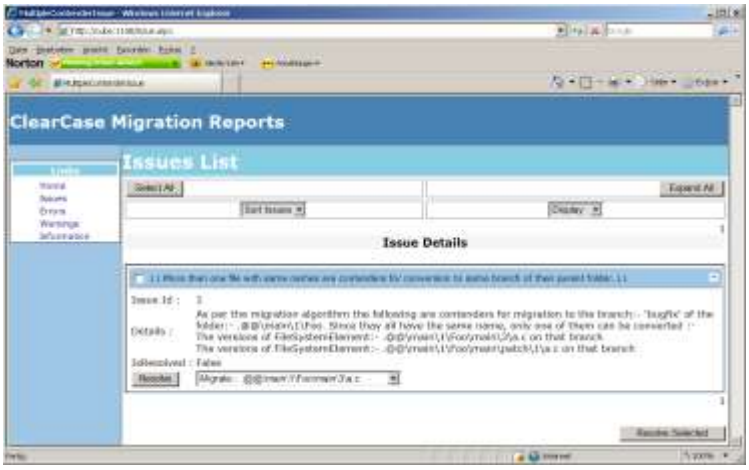


5 The issue will be solved using the reporting website, once this is completed the migration may be resumed by selecting the 'Next' button.

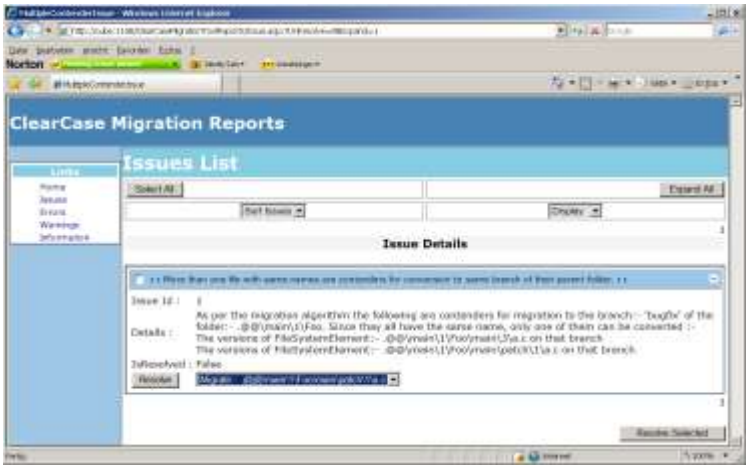
6 To resolve the issue, browse to the reporting website, select the Migration Name and the Issues link:



7 The issues are listed as follows:



8 The user is required to choose which element is to be migrated:

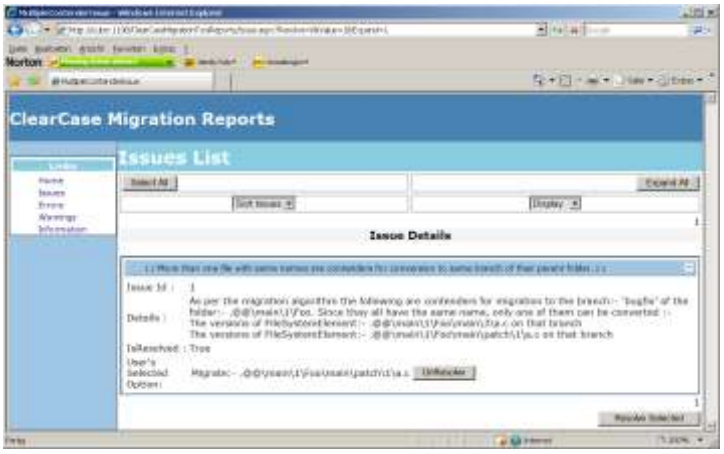


9 The possibilities listed in the Resolve drop down menu list correspond to the elements listed and explained in Chapter Technical Example:

- BranchFooMain
- BranchFooPatch

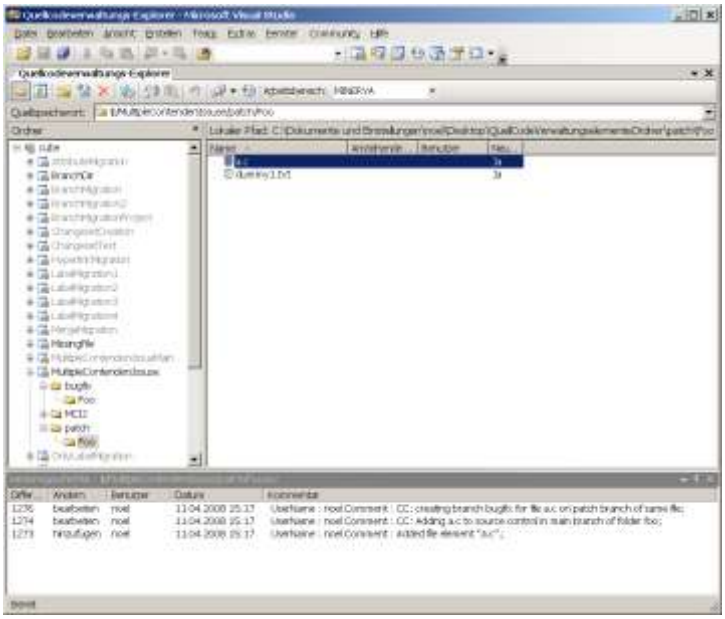
Select main\patch\1\a.c (BranchFooPatch) and the Resolve button

10 The multiple contender issue has been resolved, it is now possible to continue with the migration:

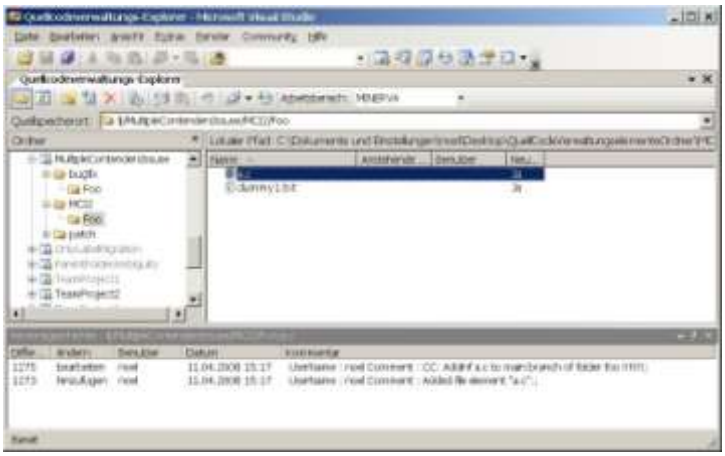


11 Return to the Migration Tool GUI and select the 'Next' Button. The bugfix branch of Foo\main\3\a.c element and all the hierarchy below it will be ignored for the migration.

The element *Foo\main\patch\1\a.c* is successfully migrated:



12 The bugfix branch of *Foo\main\3\a.c* is ignored. However the main branch of this element is migrated:



5.4.4 Migration Results Example Conclusion

The migration requires input from the user. The user decides which one of the multiple contender elements branches is to be migrated. The migration tool will ignore the other element(s) branches. The entire branching hierarchy below the ignored element(s) will also be ignored. This branch(es) of the element will have to be manually migrated at a later stage

5.5 Migrating from ClearCase and Parent Folder Ambiguity Issue

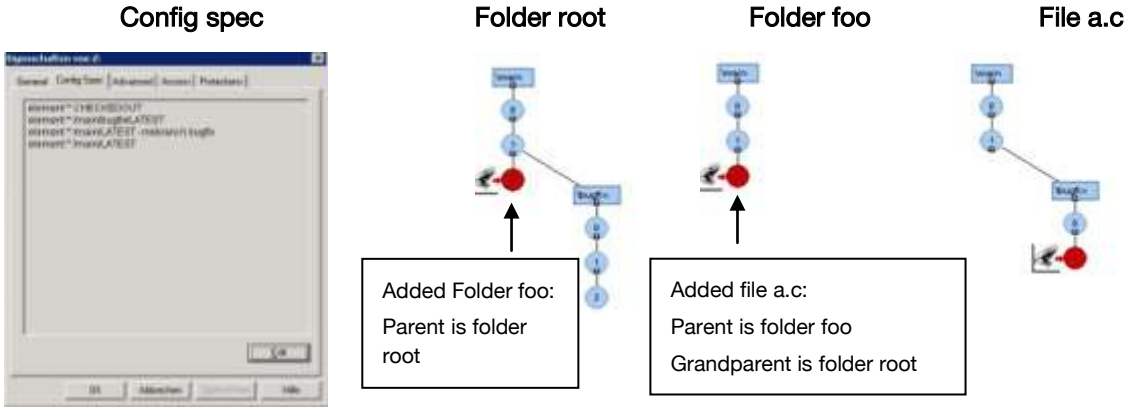
Parent Folder Ambiguity occurs when a file / folder has a parent folder that shares an identical name with another folder added on a different branch of the grandparent.

In the example below:

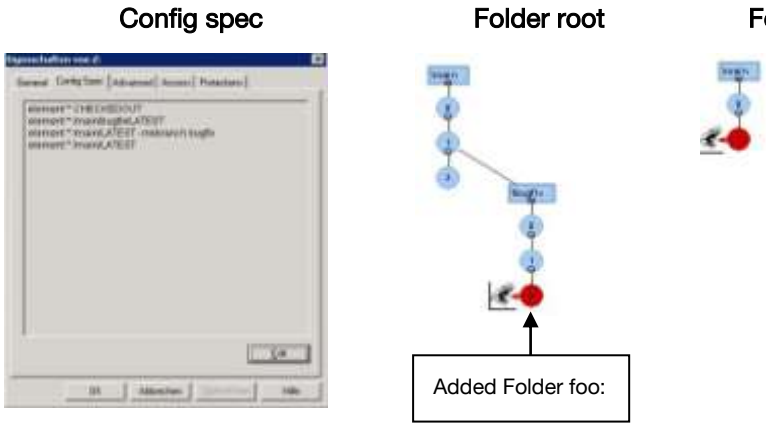
1. The file a.c has been added to the folder foo, folder foo is the parent of file a.c. This folder foo was in turn added in the main branch of folder root, therefore root is the grandparent of file a.c.
2. A different folder named foo was added to the bugfix branch of the folder root (grandparent of file a.c).

5.5.1 Technical Example

ClearCase Scenario - Folder foo added in main branch of folder root; File a.c added in main branch of folder foo:



ClearCase Scenario empty Folder foo added in bugfix branch of folder root:



Scenario Summary: Two different elements identically named foo are present in ClearCase.

5.5.2 Expected Migration Results

Two different folders named foo will be created in TFS:

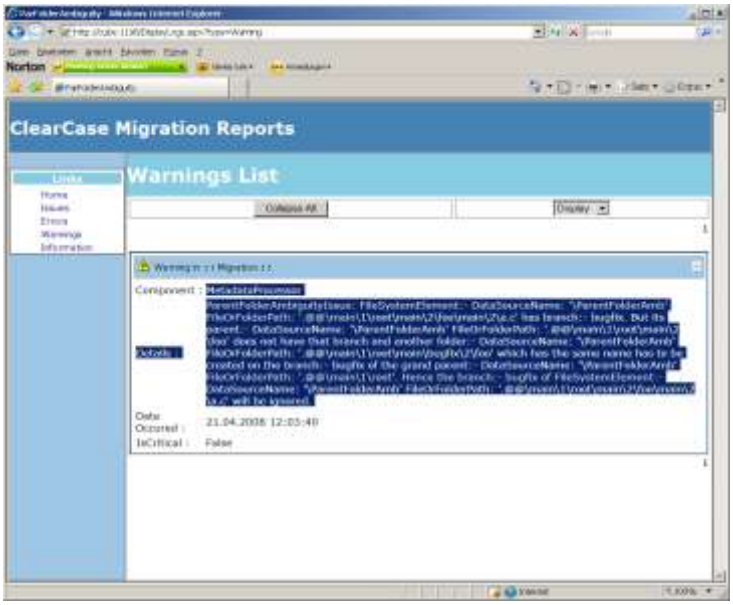
1. Folder foo for the main branch of the folder root
2. Folder foo for the bugfix branch of the folder root

The bugfix branch of the root folder is migrated to a folder named bugfix in TFS. The second of the foo folders above is migrated to this location.

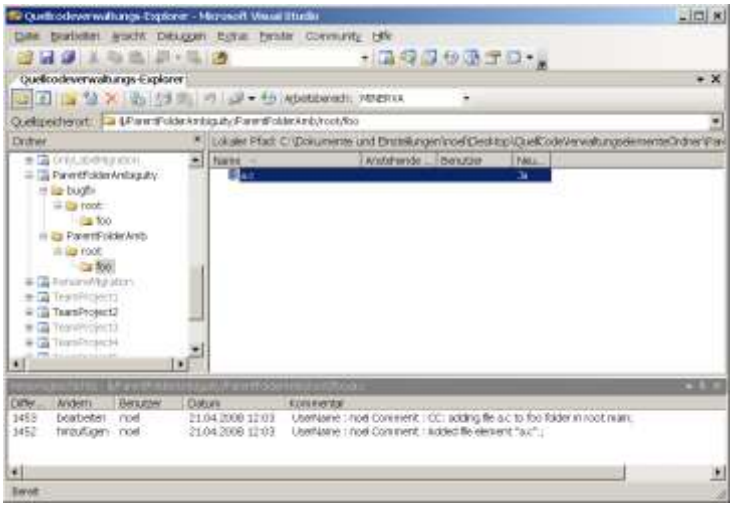
The Migration Tool encounters a Parent Folder Ambiguity Issue when it tries to migrate the bugfix branch of the file a.c – it would be incorrect to migrate the bugfix branch of a.c to the foo folder created in the bugfix folder in TFS (the second of the foo folders above), therefore the Migration Tool will ignore the bugfix branch of the file a.c, this branch of the a.c file may be manually migrated at a later stage.

5.5.3 Migration Results and Parent Folder Ambiguity

1 During the migration the Migration Tool recognizes the parent folder ambiguity that is associated with file a.c. This information is made available in the report that the user may use to identify the candidates for a manual migration at a later stage:



2 The bugfix branch of file a.c is ignored during the migration, only the main branch of this file is migrated to TFS ‘root/foo’ folder:



5.5.4 Migration Results Example Conclusion

Whenever the migration encounters a branch of a file that possesses a parent folder ambiguity issue, then this branch of the file will be ignored for the migration. The information regarding the ignored branch is available on the report website for the migration.

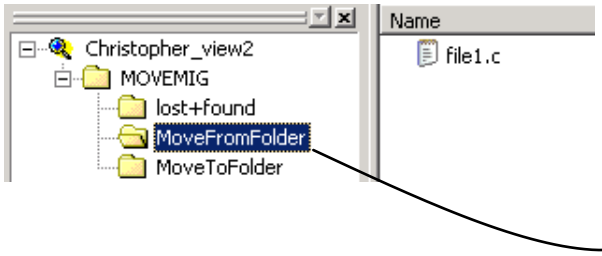
5.6 Migration Tool and Move Operations (Folder Versioning)

5.6.1 Technical Example

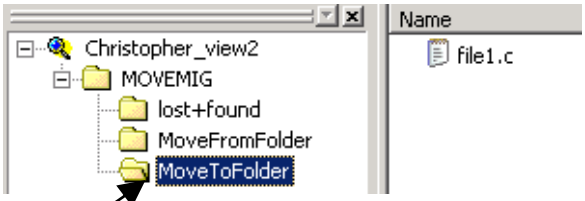
5.6.2 Migration Scenario

A file which has a branch of type ‘bugfix’ attached to it is moved from folder ‘MoveFromFolder’ into folder ‘MoveToFolder’. The VOB is then migrated to TFS.

ClearCase directory structure before Move:



ClearCase directory structure after Move:



The move operation results in the creation of two new folder versions in ClearCase, one for the source folder (MoveFromFolder) and one for the target folder (MoveToFolder).

5.6.3 Expected Migration Results

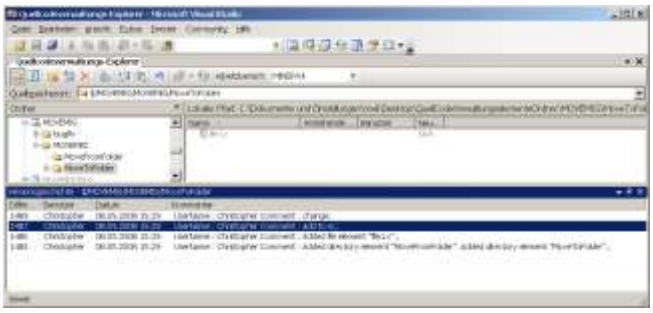
The following is expected in TFS post migration:

- The version history of the file file1.c will be correctly migrated from ClearCase
- The file and version history contents of the MoveFromFolder and the MoveToFolder are correctly migrated to TFS

5.6.4 Migration Results

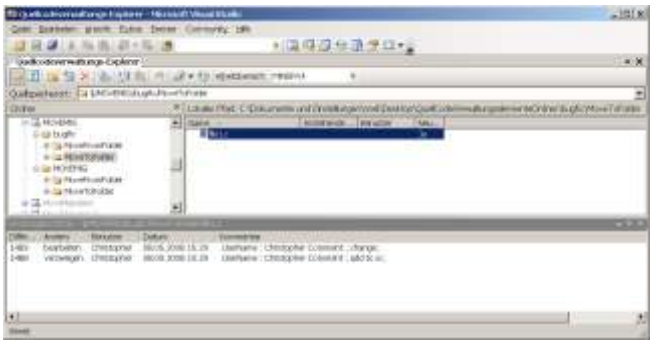
Migration complete, present in the in the Migration report is the following uncritical Error:

The complete version history of the file file1.c is correctly migrated to TFS.

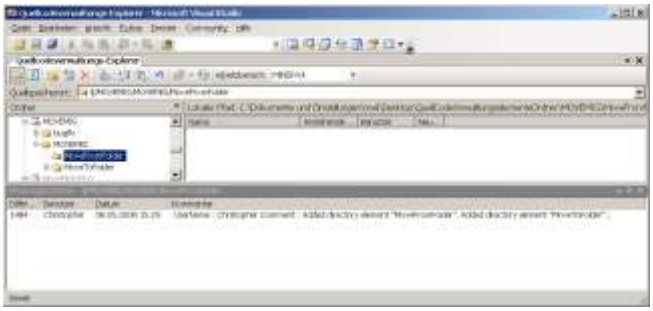


The Migration Tool does not perform a move operation - the tool migrates ***all versions*** of the moved file (file1.c) to its final location in TFS(MoveToFolder in bugfix and in MOVEMIG folders):

all versions: means the versions that were present in ClearCase in the MoveFromFolder and in the MoveToFolder



The source directory in TFS (MoveFromFolder in both MOVEMIG and bugfix folders in TFS) does not contain any version history of the file file1.c.:



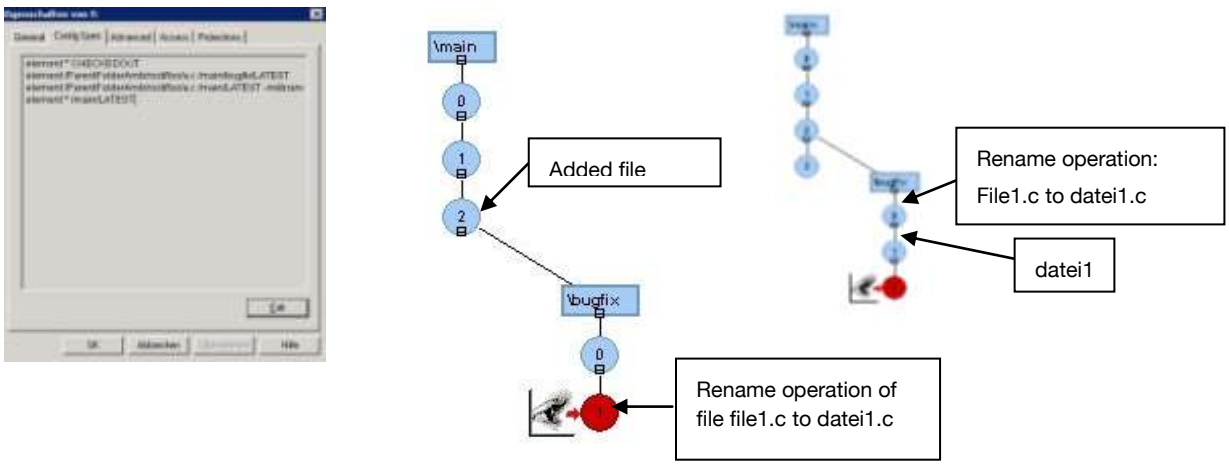
The folder version contents as they are present in ClearCase are not migrated to TFS. In the case of a move file operation, the file and its complete version history is migrated to the folder that is its final location in TFS (MoveToFolder in MOVEMIG and in bugfix folders). The versions of the file that are located in the source folder in ClearCase (MoveFromFolder) are not present in the corresponding folders in TFS, instead they have been migrated to the final location (MoveToFolder in both MOVEMIG and bugfix folders in TFS).

5.7 Migration Tool and Rename Operations

5.7.1 Technical Example

5.7.2 Migration Scenario

Config spec **Folder foo** **File file1.c**



A file file1.c consists of a main branch and a bugfix branch. A rename operation is performed in ClearCase (see above diagram). The VOB is then migrated to TFS.

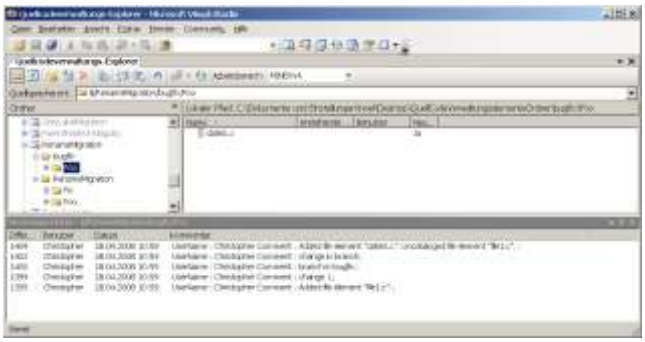
5.7.3 Expected Migration Results

The following is expected in TFS post migration:

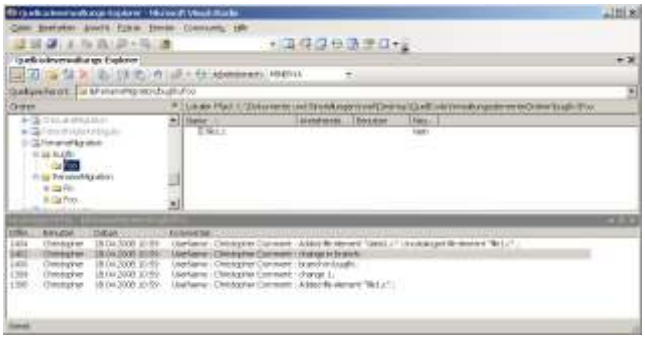
- The version history of the renamed file will be correctly migrated from ClearCase to TFS

5.7.4 Migration Results

The complete history of the latest versions of both the folder and the file affected by the name change is successfully migrated to TFS:



When calling the previous version of folder foo, the original file name version is available in TFS:



5.8 Migration Tool and Handling of Delete (CC:rmname)

5.8.1 Technical Example 1

5.8.2 Migration Scenario

- A file called file1.c is deleted in branch ‚bugfix' using the ClearCase rmname command
- The VOB is then migrated to TFS

5.8.3 Expected Migration Results

The following directory structure is expected to be created:

- The file history and older versions of the file file1.c on 'bugfix' branch will be migrated to folder 'bugfix'.
- The deleted version will not be migrated.
- All versions of the file file1.c on 'main' branch will be migrated

5.8.4 Migration Results

After the migration the structure in main and the "bugfix" branch are correct. All information regarding the delete operation in ClearCase is correctly migrated to TFS.

5.8.5 Technical Example 2

5.8.6 Migration Scenario

- A file file1.c is deleted on branch ‚bugfix' using the ClearCase 'rmname' command
- A file file1.c is created on branch ‚bugfix'

- The VOB is then migrated to TFS

5.8.7 Expected Migration Results

- It should be possible to trace the deleted and new created version of the file1.c on the ‘bugfix’ branch.

5.8.8 Migration Results

The complete version history of the file file1.c is correctly migrated to TFS.

6 User Account Management

6.1 ClearCase – User Account Management

The user management in ClearCase is handled across the Windows/Unix group affiliation. There is no user management activities performed within ClearCase. The users are registered in the relevant domain group or groups by the domain administrator. ClearCase access control is based upon this domain group membership. The ClearCase administrator workload is negligible.

6.2 Team Foundation Server – User Account Management

The user management is TFS is a combination of Active Directory and Team Foundation affiliation.

Users are initially created / defined in the Active Directory, this is normally performed by the domain administrator(s). Groups are created / defined in TFS, this is normally performed by the TFS administrator(s).

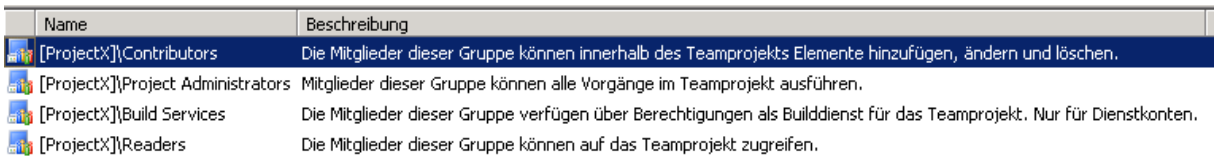
TFS user account management is based on a group concept - within TFS, users out of the Active Directory are then assigned to the relevant or appropriate TFS group(s).

Team Foundation security is user and group based. Team Foundation Server, Windows SharePoint Services and SQL Server Reporting Services all maintain their own information about groups, users and permissions.

When a project is created in Team Foundation Server, each project has three default groups, to which users and groups can be assigned:

- Project Administrators
- Contributors
- Readers

Additionally, the Team Foundation Administrators group appears as a group in every project. Users and groups may be added to these default groups. It is also possible to create custom groups and assign those groups the permissions appropriate to the business role each group represents.



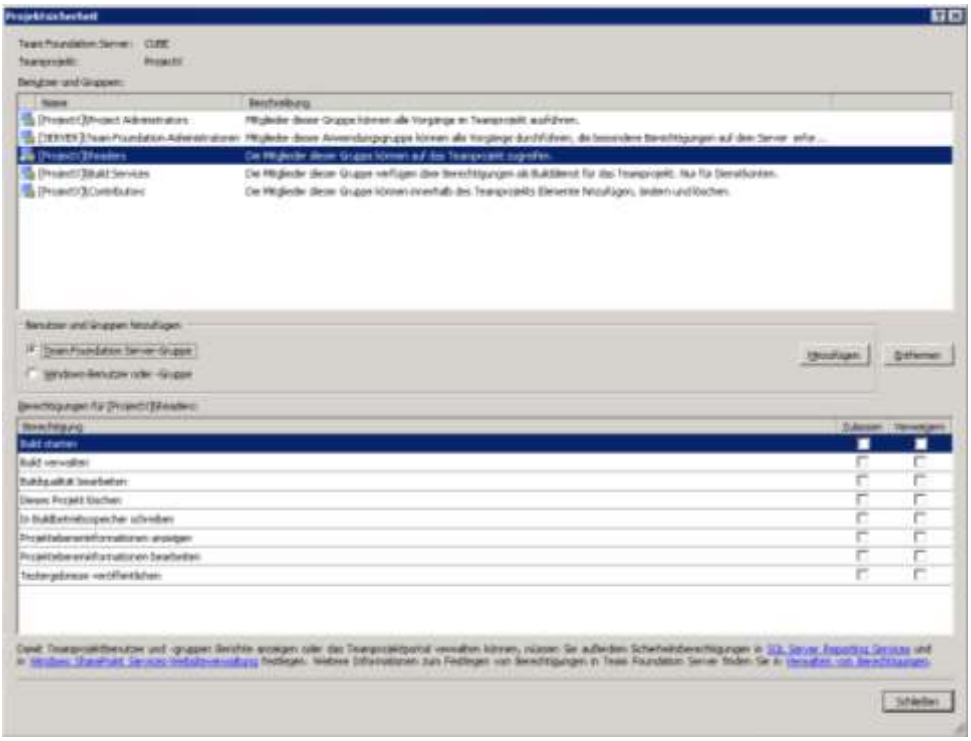
6.2.1 Security Concept

Rights are assigned to the various groups in TFS, it is important to ensure that only the groups that require the rights are actually assigned these rights.

The assigned rights (to data and functions) should allow the group members to execute their tasks, nothing more, nothing less.

Data protection can be achieved through limiting the group access. The default or standard user groups that are created during the installation of Team Foundation Server are designed to suit the security requirements of most organizations.

In cases where an organization has special or specialized security requirements, it remains possible to alter the existing security groups or create new security groups.



It is generally advisable to avoid the direct creation of users in TFS due to the fact that is time-consuming to administer the user access rights for a large number of users from within TFS. If TFS is used for managing the user accounts, then it is recommended to use user groups. It is then possible to add or remove users to and from these groups as necessary.

As a general rule the user management should be based on the existing windows domain user accounts.

6.2.2 Standard roles and security

The following is an overview of the default user groups in TFS:

- Team Foundation-Administrator
- Teamprojectadministrator
- Contributors
  Project team members with write access
- Readers
  Project team members with read access
- Build Services
  Project team members with build rights

7 Migration Roadmap

	Assess	Define	Prepare	Migrate	Operate
Practices	- Identify current configuration management practice - Identify the content that is to be migrated - Identify future configuration management standards that the customer wants	- Identify the sources that are to be migrated - Users and user rights that are to be migrated - Identify any problematic source control features (Branching ambiguities, parent folder ambiguities, evil twins)	- Install Migration Tool - Ensure that privileges are in place for the execution - Ensure that the sources to be migrated are buildable in the ClearCase environment	- Perform the migration - Configure the users and their rights in TFS	- Analyze the migration and branching issues - Perform manual migration - Coach Best Practices in Configuration Management for the Team Foundation Server Environment
Inputs	- Configuration Management Team - Customer Contact Person	- Source Control Management Team - User Management Team	- Source Control Management Team - Customer contact person - Database Admin - Network Admin	--	- Configuration Management Team
Outputs	- Existing Configuration Management Overview - Migration Content Overview (ClearCase Projects, branches, labels etc.) - Future Configuration Management Best Practices	- Sources, versions, labels and branches to be migrated - List of potential migration issues - List of users and their rights	- Username and password for the execution of the migration - Timeframe for the migration - Successful build of the sources in the ClearCase environment	- A Team Foundation Server team project featuring the ClearCase sources - A Team Foundation Server user configuration - Listing of migration issues (Use the Migration Tool Web Interface)	- Listing of sources that are to be manually migrated - Employee 'buy-in' to Team Foundation Server - Manually migrated sources are present in TFS
Test	--	--	- Analyze build results in the ClearCase environment - Perform test migration to ensure Migration Tool, TFS, ClearCase and user privileges are OK	- Perform some actions with a user account (check-out, check-in, work-item creation..)	- Perform a build in TFS, compare the results with the ClearCase build

8 Appendix

ID	Annex	Description
1	Comprehensive Guide to Migration Execution using Migration Tool GUI	--
2	Changeset Creation in TFS by the Migration Tool during the Migration	--