

Categorizing Migrations

What to Migrate?

A version control repository contains two distinct types of data. The first type of data is the actual content of the directories and files themselves which are given incremental revisions as changes are made to these elements. That is the most fundamental data that a migration needs to consider and appropriately handle. Most migrations are focused on the migration of this data.

The second type of data varies greatly in extent and implementation from one SCM tool to another – metadata (that is the data about the revisions or elements themselves). Simple metadata can contain the author, creation date and time, and log message for each revision of an artifact. This data is often required for migrations but not necessarily its duplication in the new system. Other common metadata that may be considered in a migration consists of branches and tags, or labels. Metadata can extend itself into attributes (for instance, name value pairs that often support process or file specifics such as line endings, keyword expansion, or build labels) or into more proprietary types of data unique to the SCM tool that uses it. The latter class of metadata is rarely migrated because it is difficult to map it from the SCM tool that uses it to any logical location within Subversion. It is not a matter of having a place to migrate it to, as Subversion properties can be used for such data, but rather its relevance and usefulness.

As noted in the Migration Planning document, many projects are best served by just moving the latest revisions and starting from that clean point. Others require several baselines to be migrated while again others need the intermediate revisions as well. Each of these may or may not need metadata migrations. Any metadata migrations will likely have some limitations on scope based on the legacy tool.

When to Migrate?

The ideal time to make changes to a project's version control environment is at the project's inception, leaving it untouched throughout the project's lifecycle. This document assumes that a drive to migrate to Subversion sometime during the project's lifecycle is desirable. For any migration, it is important to minimize the productivity impact of the transition as much as possible, which implies that migrations should happen at logical points (likely milestones or the inception of new release work) rather than at a random timeframe. This milestone based migration approach may imply that project sub-teams will migrate at different times based on the status of multiple release efforts. Phased migration is attractive in some cases, but makes the process much more complicated and costly. Sometimes, an arbitrary migration point has to be defined if a project either lacks an obvious migration point or if that point is seen as not occurring soon enough. Migration at logical breaks is the best approach if at all feasible.

What's the Impact of Migration?

Migrations may have a short-term impact on a project's schedule and productivity. There is the obvious impact of the time required to execute the migration, but also the impact of mastering a new tool as well as adapting to changes in the project's configuration management plan. Sometimes there are natural schedule gaps which minimize the short term productivity loss. Still, CollabNet has found this impact to be reasonably limited and easily mitigated over time by the benefits of Subversion. Key to minimizing the impact is to communicate migration planning and execution information to the entire team to facilitate migration scheduling as well as to appropriately enable users through training.

How to Migrate an Enterprise?

Some organizations choose to migrate large numbers of projects in a defined period of time. This requires additional planning and coordination and will likely affect the planning and flexibility of individual project migrations. While it is still important to try to migrate projects at milestone points, other considerations must be made when large numbers of projects need to be moved. First, it is advisable to do as much of the individual project planning as possible categorizing projects through analyzing the source SCM systems ,the desired level of migration, and the size of the repository to be migrated. If possible, it is best to identify small representative projects for each of the categories to pilot the migration techniques. For each category, the "pilot" project should be migrated with lessons learned to be applied to subsequent migrations in that category.

Following the pilot, migrations should occur within a category working up from the smallest and simplest to the largest and potentially most complex migration. As migrations are executed, lessons learned should be incorporated into the migration methods, which will help tackle larger migrations and eliminate risk incrementally. If possible, it is best to work through migrations involving no more than two SCM systems in parallel.

Migration Scope - What Data?

Migrations differ in terms of data migration objectives. CollabNet classifies migration scope into either a Selective History or a Full History type.

Selective History

Selective history migrations are those in which not all revisions and metadata are migrated into Subversion. The extent of data selection defines four types of selective history migrations:

Migration with no History – Contrary to many people's intuition as well as best practices with data from other classes of tools, this is the type that best suits most projects. This does not devalue history, but keeps it within the tool where it was created. A slimmed down version of the legacy SCM tool can be kept active for the occasional

need to access historical data. In time, it may be that the data and SCM tool can both be archived for the increasingly rare time it is needed. This type of migration can be done by the project team itself using the Subversion import command to create the initial baseline from a user copy of the appropriate legacy revisions. It leaves the development of branches to the ongoing work of the team or the execution of the general configuration management plan.

- *Cost* – Cost is low as no external resources are required and the time involved is limited.
- *Level of Effort* – Effort is mostly limited to just that required to initially create the baseline.
- *Complexity* – Little to no complexity is involved with the ability to remove the baseline or repository if the wrong baseline is imported.

Migration of Key Baselines – This is basically just repeating the first migration type multiple times. It is based on the observation that intermediate revisions are rarely useful from a historical perspective, but rather, most inquiries are focused on milestone revisions. Again, a slimmed down version of the legacy SCM tool can be kept, but that requirement is more optional with this type of migration. Again, this type of migration can be done by the project team itself and leaves branch configuration to the overall project configuration management plan.

- *Cost* – Cost is low as no external resources are required and the time involved is limited.
- *Level of Effort* – Effort is mostly limited to just that required to initially create the baselines.
- *Complexity* – Little to no complexity is involved with the ability to remove baselines or repository if the wrong baselines are imported.

Migration of Key Baselines with Structure – This approach has much in common with the previous types, but also assumes the need to migrate, or create, an initial branching scheme within Subversion based on the existing scheme in the legacy tool. A little more care and effort is required to be sure that the correct branches are created at the appropriate points within the historical record as well as the appropriate places within the Subversion data structure. This approach requires careful consideration and planning of the branching scheme that will be used as well as the appropriate baselines to be migrated. This planning effort may benefit significantly from leveraging CollabNet Consulting Services' experience and know-how.

- *Cost* – Some amount of external resources may be useful at least in the planning stages of this type of migration. Cost is low as the time involved is limited.
- *Level of Effort* – Effort is only slightly more than that required in the previous migration type.
- *Complexity* – Limited complexity is involved with the ability to remove baselines or repository if the wrong baseline(s) is imported or branching is incorrect.

Migration of Key Baselines with Structure and Metadata – This approach extends the previous type one step further with the migration of some set of metadata. That metadata may be creation information (such as, author, timestamp, log message) or data like tags, or labels. This approach requires a migration tool in order to migrate the metadata correctly, introducing the potential that a tool may not migrate the desired metadata, that cost and effort may be required to extend the migration tool, or that the migration type be reconsidered. This migration type normally requires platform access to the repository to execute the tool.

- *Cost* – Some amount of external resources may be required at least at execution time of this type of migration. Cost is “medium” as the time involved is limited, although more than with the previous types.
- *Level of Effort* – Effort is more than that required to in the previous migration type, but still quite limited based on the number of baselines being migrated.
- *Complexity* – Complexity is limited to the ability of the tool to migrate the desired metadata. As noted earlier, some types of metadata are easily mapped (for instance, creation information) and some have no obvious homes. The more elaborate the metadata to migrate the more significant the complexity.

Full History

Full history migrations are those in which intermediate revisions are migrated as well as key baselines. The extent of data selection itself defines at least three types of migrations:

Full History of Partial Structure – This approach finds value in taking the full revision set from key branches, but ignores other branches entirely. This approach is taken when pruning is seen as good hygiene prior to migrating to Subversion. Ignored branches are often developer, feature, or bug fix branches where the value is only realized from the revisions in the key development, integration or production branches. This approach requires a migration tool and it requires more selective use of that tool based on a thorough migration plan.

- *Cost* – Cost can be high depending on the extent of the branches to be migrated versus those not migrated. There needs to be significant value in both migrating the data in general as well as in pruning the structure. Some amount of external resources will likely be required at least at execution time of this type of migration as well as in planning. The time involved could be extensive due to more manual effort being required.
- *Level of Effort* – Effort is more than that required for any other migration type, but it will be bounded based on the number of branches being migrated.
- *Complexity* – Complexity is pretty high based on the need for the migration tool to easily assist with the desired pruning as well as the appropriate usage of the tool through many iterations to create the desired structure within Subversion.

Full History of Complete Structure – This approach migrates the complete project data from a revision standpoint. This migration attempts to bring every historical revision into the Subversion repository.

- *Cost* – Cost can be medium to high depending on the extent of the structure to be migrated. There needs to be significant value in migrating the data. Some amount of external resources will be required at least at execution time of this type of migration as well as in planning. The time involved could be medium to high due to amount of data being migrated so costs can be high.
- *Level of Effort* – Effort is somewhat limited assuming the migration tool can migrate the full repository from a single invocation. Data issues during migration or afterwards can impact the level of effort.
- *Complexity* – Complexity is relatively high based on the need for the migration tool to appropriately handle any data issues.

Full History with Meta Data – For either of the previous two migration types, metadata migration would add complexity, cost, and complexity just as it did with the selective migration approach earlier.

- *Cost* – Significant cost can be assumed to be added for metadata migration. Time will also increase due to the increased amount of data being migrated. Both cost and time are impacted by the type and extent of metadata being considered.
- *Level of Effort* – Effort is still somewhat limited assuming the migration tool can migrate the full repository from a single invocation. As before, data issues during migration or afterwards can greatly impact the level of effort.
- *Complexity* – Complexity is high based on the need for the migration tool to appropriately handle the desired metadata types as well as any data issues.

Migration Scope - Legacy Systems

There are numerous ways to categorize migrations. Migrations could be categorized by the data repository size, repository structure, and the extent of history. It can be generally assumed that migrations get progressively more costly, time-consuming and complex as repositories are larger, have more extensive branching, and have a larger number of revisions for each element. This affect of data repository complexity will impact each of the categories below. Since the originating tool can play an important role in the scope, impact and difficulty of the migration, the following sections categorize migrations based on VC/SCM tool parameters and evaluates them against a number of migration attributes:

CVS or Stand-Alone Subversion

CVS migration falls into its own category based on broader familiarity with it as well as its role as the most common migration to Subversion being done today. It is also a potential path for other migrations that might use long-established migration tools to convert first to CVS and then to Subversion. In this case both the source and target systems are well understood and have a common model.

- *Commonality* – Since Subversion was designed to be a replacement for CVS, there is a lot of similarity in the two tool models. This allows for logical data mapping.
- *Available Tools* – CollabNet helped develop the most commonly used migration tool for this category, cvs2svn.py, and uses it frequently to convert customer repositories.
- *Complexity* – CVS migration is the most understood migration category having the highest ability to modify existing tools if required.
- *Time* – There is no way to accurately define the time required for a migration. This tool not only migrates the revisions and the metadata, but also attempts to create atomic commits based on the historical data. CollabNet has seen migrations that have taken hours (some just minutes) to those that have taken several days.
- *Recommended Migration Type* – There should be an effort to determine the need and ROI for any migration of data. If data migration seems warranted, then any of the previously discussed migration types can be pursued with full history and metadata the most commonly selected option (among those migrating data).
- *Expected Impact* – Given the commonality between CVS and Subversion, it can be expected that, with proper training, a project team will quickly (within a day to a week) be as productive with Subversion as they were with CVS and will subsequently be more productive due to Subversion's feature improvements. There also should be limited impact to the configuration management processes due to the same commonality. If a full migration was carried out, there will be no need to support a historical CVS repository. If a more limited migration approach was used, then there would be limited impact in maintaining a historical CVS repository with the potential to further limit that impact by archiving that repository and CVS when demand diminishes.

Limited Featured VC Tool

Tools that have limited functionality beyond basic version control enable fairly simple migration mappings that can easily be automated. An example of this type of migration is a Visual SourceSafe migration.

- *Commonality* – Since the source tool has a limited focus which is the basis for Subversion, there is a lot of similarity in the two tool models. This allows for logical data mapping with few exceptions.
- *Available Tools* – While some Open Source tools exist for the purpose of converting Visual SourceSafe and PVCS repositories, you may find the best approach is to first convert them to CVS and then to Subversion.
- *Complexity* – Complexity is higher than with CVS, but given the limited scope of the source tools and the Open Source nature of tools, it should be manageable if migration is required.
- *Time* – There is no way to accurately define the time required for a migration. However, most tools in this category can generate a formula over time that correlates the number of elements and the average number of revisions to the time

a conversion takes. Based on the amount of source data, migrations will likely range from hours to a few days.

- *Recommended Migration Type* – There should be an effort to determine the need and ROI for any migration of data. But if data migration seems warranted, it is best to limit the amount of data migrated. However, it is likely that a full history with metadata can be executed.
- *Expected Impact* – Given a common focus, it can be expected that, with proper training, a project team will quickly (within a couple of days to a week) be as productive with Subversion as they were with the previous tool and subsequently be more productive due to Subversion's feature improvements. There also should be some impact to the configuration management processes if the processes are adjusted to take advantage of Subversion-specific functionality. If a full migration was carried out, there will be no need to support a historical legacy repository. If a more limited migration approach was used, then there would be limited impact in maintaining a historical repository with the potential to further limit that impact by archiving that repository and legacy tool when demand diminishes.

Full Featured VC Tool

Examples of full featured VC tools are Perforce and BitKeeper.

- *Commonality* – Since the source tool still is focused on version control, which is the basis for Subversion, there is a lot of similarity in the two tool models. Distributed SCM tools, like BitKeeper, may seem to be more divergent than they really are from a migration perspective. The general commonality allows for logical data mapping with some proprietary exceptions (e.g., most commonly merge tracking).
- *Available Tools* – Some Open Source tools exist for the purpose of converting Perforce, but other tools in this category are likely best converted first to CVS and then to Subversion. This may also be the best approach for Perforce.
- *Complexity* – Complexity is higher than with CVS or limited feature version control tools, but given the focused scope of the source tools and the Open Source nature of migration tools, it may be manageable if migration is required.
- *Time* – There is no way to accurately define the time required for a migration. However, most tools in this category can generate a formula over time that correlates the number of elements and the average number of revisions to the time a conversion takes. Based on the amount of source data, migrations will likely range from hours to a few days.
- *Recommended Migration Type* – Obviously, there still needs to be an effort to truly determine the need and ROI for any migration of data, but if data migration seems warranted, it is best to limit the amount of data migrated. However, it may be possible that a full history with metadata can be executed.
- *Expected Impact* – Given a relatively common focus, it can be expected that, with proper training, a project team will quickly (within a few days to a week) be as productive with Subversion as they were with the previous tool and will subsequently be more productive due to Subversion's feature improvements.

There also should be some impact to the configuration management processes if these processes are adjusted to take advantage of Subversion-specific functionality. If a full migration was carried out, there will be no need to support a historical legacy repository. If a more limited migration approach was used, then there would be limited impact in maintaining a historical repository with the potential to further limit that impact by archiving that repository and legacy tool when demand diminishes.

Full Featured SCM Tool

Full Featured SCM Tool migration is the most difficult based on the extensive proprietary metadata and features found in most full featured SCM legacy tools. Examples of this type of tool are ClearCase, Dimensions, and CMSynergy.

- *Commonality* – Complexity is considerably higher than with the other categories of tools so complexity is relative to the migration type desired.
- *Available Tools* – Some Open Source tools exist for limited conversions of ClearCase, but not for most others.
- *Complexity* – Full Featured VC Tool migration complexity is higher than any previous category, which may prevent or greatly complicate any broad migration effort.
- *Time* – There is no way to accurately define the time required for a migration. Tools in this category will be difficult to accurately estimate and will depend upon the breadth of the data that is being migrated.
- *Recommended Migration Type* – True history migration types are highly discouraged for this category based on cost and complexity. If pursued, the project team must have realistic goals for what metadata should be migrated.
- *Expected Impact* – Given a broader focus in the legacy tool, it can be expected that, with proper training, a project team will soon (no more than one to two weeks) be as productive with Subversion as they were with the previous tool with the potential to be more productive due to the feature differences. There will likely be impact to the configuration management processes to adjust these processes to take advantage of Subversion-specific functionality. If a full migration was carried out, there may or may not be a need to support a historical legacy repository. If a more limited migration approach was used (most likely for this category), then there would be limited impact in maintaining a historical repository with the potential to further limit that impact by archiving that repository and legacy tool when demand diminishes.

Conclusion

Migrating a project from one version control tool to another requires careful consideration of what really needs to be migrated into the new environment, what may be left in a limited legacy system, and what may not be useful at all. It also requires careful determination of when to migrate. The impact of migration on the project must be taken into account as well as how it will be accomplished. Lastly, one must consider what

scope of data will be included in the migration. This document has attempted to help you through these considerations.