

Standardizing on Perforce

Karen Brooks, VP Operations, WSI Corporation

RJ Allen, Senior Systems Administrator, WSI Corporation

Neal Firth, PCP/ PCT, SageRight, Inc.

WSI Corporation is the world's leading provider of weather-driven business solutions for clients that include CNN, FOX, NBC, American Airlines, Delta, and FedEx. Their innovative products, services and software satisfy the professional weather needs of the most demanding media, aviation, energy trading and utility customers in the world.

The Need for Standardization

Before a Standard

WSI is organized in business units with each business unit delivering products and services to its customer base. This organizational structure coupled with WSI's broad breath of offerings resulted in a heterogeneous development environment. In early 2006 WSI found themselves with code residing in several different source control systems - CVS, VSS, ClearCase, and SCCS. Each SCM system had several repositories. Most repositories had been in use for over 5 years - with one SCCS repository having almost 25 years of history.

Several of the existing systems had reached practical operational limits and needed to be either replaced or upgraded. VSS was having issues with branching, database size, corruptions, and use in a distributed environment. ClearCase was an end-of-life version, needed significant hardware enhancements to restore performance, and was proving problematic to use in a distributed environment.

Each business unit had evolved a set of practices that were strongly influenced by the capabilities and characteristics of the SCM tool they used. Different capabilities meant that there was limited commonality to the vocabulary used to describe practices. Gains in efficiency and productivity were difficult to share between the business units. There was limited opportunity to share problem solutions or source code across business units.

There was also a need in each business unit to be more agile in its project staffing requirements. WSI wanted to implement flexible staffing through outsourcing. The current source code systems did not provide robust remote access capabilities. This limited WSI's ability to become more agile in how it staffed short and long term projects. The system selected needed to provide a robust capability for providing remote access to source code.

Investing in a robust, flexible source code management system would allow WSI to implement company-wide build automation tools, share standard software functionality across business units and reduce overall cost of ownership through reducing internal support overhead.

Deciding on a Standard

WSI president Mark Gildersleeve chose Karen Brooks to head a corporate initiative to increase productivity and reduce cost of ownership by standardizing source control and defect tracking practices throughout the company.



WSI went through a review cycle of several SCM and bug tracking systems. The three key criteria in making the selection included integration between the two systems, tools available to manage the overall development process, and cost of ownership (license and maintenance). Pragmatic was implementing a bridge between the two products and would provide WSI with the level of SCM and bug tracking integration they desired. Additionally Pragmatic offered functionality to automate the development process management and flexible naming conventions that allowed WSI to accommodate its many development projects across its business units.

The final step in the decision process was to find a consultant team who could deal with the complexities of the WSI development environment. WSI selected SageRight because of their breadth of experience, knowledge of engineering best practices and understanding the challenges of the task at hand. Neal Firth, Founder, met with several members of the WSI development team and demonstrated his understanding of the challenges of this project. Neal, working with the WSI development team, put a plan in place to migrate all development teams to a centralized SCM and bug tracking system. The project plan included definition of tasks, end user training, migration, integration and automated build tools.

Migration Planning

In an ideal world, a migration would transparently convert bits and developers would simply walk in one day to find that they were using Perforce. In the real world there is a limited amount of time to consider an almost infinite number of factors. A hands-on approach with expert guidance reduced theory to practice in a manageable timeframe.

Challenges from the Beginning

Early on, these issues were established as having a potentially critical impact on the migration process:

- Limited familiarity with Perforce by WSI personnel.
- Maintaining an ongoing development environment.
- Different windows of opportunity for migration due to ongoing project development.
- Early trials found that some repositories took in excess of 48 hours to migrate. This time issue was aggravated by some development projects requiring the information from 6 or more of these repositories.
- There was a large repository of legacy defect information. However, with many hand-entered values, the validity of the information as well as the relationship between defects and artifacts under source control was frequently in question.

A set of side-effect challenges were created by corporate growth and changes to the WSI computing infrastructure during the migration period. In particular, increased network traffic and changes to the characteristics of the corporate DMZ all impacted the process by extending the time it took to accomplish tasks that involved the corporate computing infrastructure.

Individual Roles

There were three key components driving the project; business unit advocates, a project champion and SageRight. A steering committee for the project was formed as it wasn't practical for any one individual to have a complete understanding of the development, support, and

delivery practices used by each of the business units. This committee consisted of one or more advocates from each business unit and a project champion. The business unit advocates were volunteers and the project champion was selected by the President of WSI. This approach insured that the needs of each business unit were represented in the overall migration plan. The project champion had the internal visibility to quickly resolve any issues, arbitrate disagreements or remove any project road blocks.

Though a series of group and individual meetings, the steering committee made decisions concerning the migration of such items as legacy artifacts and how the migration output would be validated. The ease of each business unit's migration was directly impacted by the amount of time invested by the business unit's advocate. For those business units that had an advocate who was consistently involved in the steering committee planning sessions their migrations tended to proceed with minimal problems. Even for those business unit advocates whose time was limited the migrations proceeded relatively problem free due to SageRight's efforts in maintaining up to date written plans.

In typical projects where impact spans multiple organizations there is generally atrophy and "red tape" when it comes to making decisions. This project didn't have this experience because the business unit advocates were volunteers who felt an ownership of the process. This coupled with a clear understanding of the end goal resulted in a limited need for the project champion to intercede. There was another interesting by-product of the steering committee meetings. The discussions at the meetings often expanded to include engineering processes and suggestions in how to improve WSI's overall engineering infrastructure and processes.

SageRight's role was to provide the vision of how to translate legacy representations into Perforce representations. In addition to guiding the overall project, SageRight also worked closely with the project champion to identify conflicting organizational requirements.

How and What to Migrate

No advocate had significant Perforce experience. Early in the process, SageRight's Perforce Certified Trainer familiarized key team members with Perforce through focused user and administrator training. Specific emphasis was placed on workspace management, branching, and the use of changelists during this training period.

After the familiarization process, each team advocate worked with SageRight to establish the best strategy for establishing how and what to migrate. This process generally progressed in this sequence:

- Assess the value of migration artifacts such as source histories, branch histories, and labels.
- Assess legacy support needs. Adjust artifact requirements as necessary.
- Assess impacts of previous decisions on specific development, build, and release tools. Adjust as necessary.
- Assess impacts on delivery mechanisms. Adjust as necessary.
- Establish a migration validation procedure. Adjust as necessary.
- Establish a "go live with Perforce" strategy so that the migration would not impact ongoing development efforts.

The most common adjustment was to reduce the list of “critical” artifacts. In a post-migration environment many previously “critical” artifacts were found to be specific to the tool in use and thus had limited future value. Intermediate labels, labels as artificial branches, and shared files were the artifacts most commonly removed from the migration list. There was also a common issue of validating the integrity of historic information. In general, the decision was not to migrate potentially erroneous historic information.

Target Perforce Environment

The central Perforce server configuration is two Dell PowerEdge 2850s with Dual Xeon 3.2GHz processors, 8 GB of memory, and 4 73 GB SCSI drives arranged as two RAID1 arrays. One server provides production level user support. The other server acts as a warm spare and test environment. The warm spare duplicates the production server on a daily basis. Trigger and procedural changes are evaluated on the spare server prior to deployment on the primary server.

Proxy servers are used at overseas development sites. Proxy servers are HP ProLiant DL320G5s with Dual Core Xeon 2.13 GHz processors, 2 GB of memory, and 2 250 GB SATA drives configured as a RAID 0 array. The Proxy servers provide a significant workspace access advantage as several of the repositories contain 1000s of large graphical artifacts

All servers are running RedHat Linux ES 4.0 (2.6.x kernel)

Migration Mechanics

Original planning was based on the observation that the practical value of an historic artifact version diminishes with relatively direct relationships between its age, its frequency of use, and its frequency of change. For example, previous versions of a commonly used artifact that hasn’t changed in 5 years have limited current value. Likewise, if an artifact has changed many times since it’s last use, those previous versions tend to have limited current value.

With some of the core artifact histories going back 12 or more years it was clear that migrating full histories would require a long elapsed time and create a large amount of information with relatively limited usefulness. Regardless, some WSI customers have very long acceptance cycles and support is being provided for product based on code that is more than 4 years old.

Several migration approaches were considered including tip-only and converting against “snapshots” in time. A few repositories with extremely stable artifacts were converted at their tip versions. However, most repositories, required the conversion of 6 or more years of artifact history. A key factor supporting this decision was the ability to adapt existing SageRight migration tools to support incremental migrations.

Incremental capabilities eliminated most of the typical time and dependency issues associated with migration projects. Lengthy baseline migrations could be completed and validated as background activities in advance of planned “go live using Perforce” dates. Final migrations had a high probability of success and an elapsed time that was measured in hours. The different teams were able to schedule “go live” events on days of the week that best fit the dynamics of their user community. Incremental migration capabilities also afforded the teams significant scheduling flexibility because “go live” events required coordinating a relatively limited set of resources. Ultimately, several teams delayed “go live” events while one team was able to effect a transition a week early.

You Can't Convert It All – Dealing with Uncertainty

It is unrealistic to expect every historic artifact to convert. There are artifacts with no cross-system equivalent. Others do not have deterministic metadata extraction capabilities. Historic validation is problematic given a limited timeframe. And, with the long utilization histories each system was found to have some amount of corruption.

Initially, members of the user community had significant concerns about the migration. Most of these concerns were associated with the potential for not having access to full artifact histories. Three decisions significantly reduced the level of uncertainty among the user community:

1. Read-only access to the old systems is still available. In practice these older systems are almost never used, but the safety net is present.
2. Legacy builds are supported with a “disconnected-tips” strategy. This approach guarantees build source compatibility with a high likelihood that the available versions have associated historic information.
3. Current development was guaranteed tip equivalence with an extremely high probability that artifact history would be consistent with current systems.

Dealing with Case Sensitivity

Hosting the Perforce server on a Linux platform means that it would be case sensitive. Hosting on a platform that was not sensitive to character case was not an option as some Unix based products required case sensitive filenames.

The CVS and ClearCase repositories were already operating in case sensitive environments. However, while dealing with the migration of the case insensitive systems several projects were found to have unintended – and unwanted – case sensitivity issues.

Windows based clients presented several challenges. The character case of pre-migration workspace directory elements tended to be unique to individual developer workstations and different from the VSS project versions of the same elements. Visual Studio solution and project files tended to have a hybrid of the known directory and file element character case. Moreover, Visual Studio control files tended to have file references with significant levels of indirection in pathnames.

Using the most recent VSS version of character case as a standard for migration allowed creation of a case normalization script. This script evaluates the various Visual Studio control files and proposes appropriate changes. There can be very bad consequences if these control files are modified incorrectly so the overhead of user review prior to update was a prudent, though time consuming overhead. Finally, users had to re-create workspace environments post-migration to assure consistency of character case within pathnames. A “check my system for consistency with expectations” script is being developed as part of an ongoing effort to automate validation of user environments.

A case consistency validation trigger was created and installed to assure long-term consistency. The trigger allows each depot to determine whether or not case makes path elements unique. The trigger also provides control over where in the depot structure users can create files and branches.

Dealing with Web Projects

In addition to case sensitivity issues web projects also introduced the complexity of replicating VSS project mappings. This was handled with a custom script that extracts VSS project mapping information and translates it into a workspace view.

As a standard practice, each web project uses a master template workspace to assure consistency between clients. A “validate my workspace view against the master template view” script is being developed as part of an ongoing effort to automate validation of user environments.

Prior to the Perforce transition, VSS project mappings made moving web project environments between hosts a complicated and error prone process. Differentiating between required placements, inconsequential placements, and erroneous placements was difficult, tedious, and error prone. Once a web project workspace view is established, placement becomes deterministic using Perforce.

Disconnected-tips

Validating that converted historic artifacts yield equivalent workspaces when extracted from Perforce can be a daunting, time consuming task. Source refactoring, differences in SCM capabilities, glitches in the migration process, and trust in the comparison baseline can lead to issues that all take time to resolve. Due to the overhead of resolving infrastructure dependencies, some original SCM systems were used for short term legacy support. This is not, however, a desirable long-term practice as the propagation of changes between systems is problematic at best. Regardless, users need to have confidence that the post-migration environment supports maintenance of legacy codelines.

SageRight introduced WSI to the concept of disconnected-tips to resolve this dilemma. In effect, you create a branch tip by managing the branch as if you were working disconnected. The procedural details of creating disconnected tips are outlined in Technical Note NOTE002 – working disconnected, and NOTE015 – vendor branches. There are two key observations for disconnected-tips:

- Create a baseline version of a Perforce legacy support branch by integrating from the most appropriate sources. Source selection is important to the usability of historic information, but not to the ultimate usability of the branch.
- Overwrite the Perforce workspace with the desired tip versions from the legacy SCM system. If tip versions are missing or the SCM system is experiencing corruptions other sources may be used. Be sure to remove any files from the structure that are used to support the legacy SCM system.

In the best case, the new tips are identical to the branched sources and no changes are required. More realistic is to expect that a certain number of files are different. Regardless, you now have a working branch that replicates a legacy environment and integrates with an existing depot structure.

How Long Does Migration Take?

The total number of file versions converted tends to be the most significant factor in estimating the time it will take to migrate a repository. Establishing baseline performance numbers from a small test conversion takes into account most processor and network overhead.

For VSS and CVS the conversion time was directly proportional to the number of file versions being migrated. The one wild card to these estimates is very large text files (500K+) with a significant number (200+) of versions. The presence of these files tends to add about 20% to the migration time.

For ClearCase, the number of labels on a VOB, and the number of labels on an individual artifact version, was a significant factor for both metadata extraction and migration. VOBs with fewer than 50 labels converted about 5 times faster than VOBs with 1,000 or more labels. A similar performance difference was encountered with metadata extraction unless you included label information as part of the metadata. If label information was included as part of the extraction process then VOBs with large numbers of labels extracted 20 to 50 times slower than extractions of VOBs with small numbers of labels.

SCM System	Number of Repositories	<i>Historic Metadata Extract</i>	Number of File Versions	Total Elapsed Conversion Time
VSS	3 databases	1 hour	150,000	35 hours
CVS	15 repositories	2 hours	176,000	25 hours
ClearCase	29 VOBs	30 hours	200,000	240 hours

Conversion Characteristics

The incremental time between a baseline migration and the “go live” date was typically 7 to 14 days. Incremental metadata extraction and migration took approximately 15 minutes per repository.

Dealing with Legacy SCM Systems

Each of the legacy SCM systems presented unique challenges. And of course, the need to migrate exposed all of those “we’ll fix/ cleanup/ refactor when we have time” projects that hadn’t yet made it to a “to be done” schedule.

Refactoring Source Structures

Some of the project teams wanted to use the migration as an opportunity to refactor their source structure. And some of the repositories required refactoring to adapt to a branching structure. The appropriate refactoring mechanism was a common discussion topic.

Fundamentally there are three ways to refactor as part of a migration: you can refactor in the legacy system before the migration, you can refactor as an integral part of the migration, or you can migrate then refactor within Perforce. Even if refactoring in the legacy system is possible, it introduces a significant amount of uncertainty into the validation process and leaves the legacy system in a state that is unfamiliar to users. During migration processing, anything more than the introduction of a straightforward branching structure breaks the relationship between sources and complicates validation. Migrating effectively equivalent structures then refactoring in Perforce maintains a relationship between sources and simplifies validation at the cost of an additional processing step.

No one technique worked for all projects so several approaches were used. For projects retaining an existing branching structure migration was a one-into-one process. Projects without branching

were migrated directly into a mainline branch. Projects requiring significant source structure refactoring were migrated into a migration branch. This migration branch was used as the source of refactored mainlines after the migration had been validated.

Common Decisions

There are several decisions that need to be made regardless of the legacy SCM system involved:

- File types - where a specific file type could be trusted and had a Perforce equivalent the migration script enforced it. Otherwise, a typemap entry or default determination was used. After much debate, the CVS sources were brought over without keyword expansion. And some of the “trusted” file types turned out to have been applied inconsistently over time so a “type check” and cleanup of the final conversion was still required.
- Checked out files – there are a few people that “forgot” there was a migration. But mostly, migrations forced decisions about files that have been checked out long ago and subsequently forgotten.
- Labels – ultimately, only a limited set of label information was migrated. Of the 1000s of labels across the many repositories very few proved to have significant post-migration value. Validation was part of the problem. But more significant was the decision to support legacy builds using a “disconnected-tips” strategy. This decision virtually eliminated the need to migrate labels. In addition, creating labels was a separate process so labels could be established at any point in the future.
- Version translation – as part of their normal operation, the migration scripts created legacy version to Perforce mapping information. This information was used to control the re-creation of labels. There have been no post-migration requests by users to use this version translation information.
- Synthetic changelists – the CVS and VSS migrations combined checkins by the same user, with the same comment, within a specific time window into a common changelist. For support reasons, the ClearCase migration migrated each individual file version as a separate changelist. The version of the migrated ClearCase artifact was added to the submit description.
- Large (500K+) text files with 100s of versions. There aren’t many of these, but they do represent a choice of efficient representation on the Perforce server. In the end, the server was able to handle everything but the 4Meg file whose “trusted” type was incorrectly set as text.
- Identifying migrated sources. There was a desire for a permanent and easily identifiable mechanism for identifying migrated sources. All migrations were performed using workspace names that started with the word “migrate” followed by a word that identified the source of the artifacts. All submit comments generated during the migration process contained text identifying their migration origin. After submit, the username and date-time of changelists were set to reflect values from the original source control system.

CVS

Two significant problems were encountered. Neither of these problems manifested themselves during normal development activities. They were only significant to a migration process.

In several cases repository files were found that contained extra random bytes at the end of the file. Histories would report correctly, but CVS update commands would be confused by these extra bytes and terminate when the earliest version of the file was accessed. There were also a large number of labels within the repository files that referenced file versions that did not exist.

Although there is no “smoking gun”, both of these problems appear to have originated several years ago when a developer was experimenting with using a CVS interface that was not based on the standard distribution used by the other developers.

VSS

The challenges with VSS migrations are well known, well documented, and legend. Regardless, the ability to address two key problem areas were key to a successful migration at WSI.

The ability for the migration processing to work-around missing and corrupt files was critical. At several points corruptions were encountered in the middle of important project structures so isolation was not an option. Corruptions were exposed during metadata extraction and during the migration. Migration corruptions were associated with missing repository files and to a project structure that had been restored with a name that only differed from an existing structure by character case.

Finally, the many possible ways to change history with VSS was a concern to the users. Standardizing on the decision to migrate to tip equivalence proved to be a significant simplification to both expectations and validations. Some of the history may prove “interesting” when viewed through Perforce, but the read-only VSS copy is always available as a reference.

ClearCase

Mechanically, the biggest issues were dealing with directory versioning, fundamental differences in workspace management, fundamental differences in branching, and ClearCase evil twins.

Directory versioning impacts qualified name resolution, deleted files, renamed files, and symbolic links. All of the information is there. It’s just a bit tricky to access and associate with a time of occurrence.

Developer workspace views of branches were typically backed with main/LATEST configurations. A significant number of developer discussions centered on recreating a branch relative historic definition of LATEST. In the end, it was determined that understanding the relationship of branch unique files to a mainline was more important than actually creating a simulation of a dynamic ClearCase branch using Perforce’s static model. Prior to the migration from ClearCase to Perforce the ClearCase mainline was brought up to date so that the migrated mainline could be used to create development branches. The migration created a full historic mainline with branches populated with only branch unique file versions. With this migration technique all of the file information required for disconnected-tips was available within Perforce so legacy support could be provided.

Several instances of Evil Twins (and Quads) were identified within the ClearCase repositories. No one set of rules could be automatically applied to select between the evil threads. Ultimately combining a best version config spec with a convert to tip-equivalence strategy was used to select migration artifact versions. And the ClearCase repositories were still available if there was a need to access the Evil threads.

SCCS

One internal tool group had been using SCCS for over 20 years to track various project artifacts. With support for only the most basic of SCM technologies the migration of SCCS sources is mostly a matter of recreating a sequence of file versions.

With multiple developers using individual copies of the “master” repository, selecting the appropriate version sequence can be very challenging. For some artifacts the best choice of tip version has proven to be a deployed version. The disconnected-tip technique has been successfully applied for establishing “best choice” histories with a usable tip version.

Perforce

A new development project was initiated at WSI after the decision to move to Perforce was made but before the primary Perforce server was available. From the outset, this project used Perforce. However, they were not the first project to migrate to the primary server. They also used a Windows based server prior to their migration.

A Perforce to Perforce migration tool was created. One upside to this tool is that it effectively eliminated the need to deal with case sensitivity issues. And, although most activities were straightforward file tampering prior to submits on the source system made comparison of integrations and file histories somewhat problematic. An additional upside to this tool is that it allows SCCS based projects to prototype into an offline Perforce environment and then use this tool to migrate the most appropriate prototype to the production Perforce environment.

After the Migration

This section provides an overview of some specific WSI deployment choices and plans for the future.

Depots

Each product area was allocated a separate depot. This was done to facilitate management activities and to reduce the general “clutter” that developers would need to deal with.

A special purpose depot called PLAYTIME was created to allow users to experiment with Perforce while still having access to their primary development sources. The contents of PLAYTIME are transient and subject to being obliterated at frequent intervals.

Branching Scenarios

Branching use cases were established for each of the projects. Branching was based on the mainline model using a baseline protocol. Several common scenarios were established to deal with differences in development, delivery, and support. The scenarios deal with:

- Differences in the resolution and propagation of bugs caused by product delivery mechanisms.
- Propagation of bug fixes and features between general product development and special delivery products.
- Selection of final product feature configurations

- Propagation of features between mainline development and special projects that maintain a different release cycle.

Automation and Triggers

Several scripts have been created for supporting ongoing user environments. Most are oriented toward identifying the inevitable user-created inconsistencies that occur in any dynamic growing environment. These are outlined in other parts of this paper.

In addition to the case validation trigger described elsewhere in this paper a trigger was created for integrating external LDAP based authorization.

As processes evolve, future triggers for managing job and changelist relationships, conformance to workspace mapping standards, and conformance to naming conventions are planned.

Corporate Web Site

Prior to migration the corporate web site used VSS shadow folders as a push technology. While this works in general, it is very hard to prototype web sites or recover from the inevitable push of a “bad” version. After migration a combination of branching structures and automated processes are going to replace the VSS shadow folders. The goal is to add validation and deterministic rollback to a push based deployment.

Defect Tracking and Team Collaboration

The final phase of the migration process is to integrate Perforce with a common defect tracking system using jobs. Currently, several ClearCase based projects use DDTS. Various factors make DDTS inappropriate for wide-scale deployment in a WSI development environment based on Perforce.

Software Planner from Pragmatic Software has been selected to provide a corporate-wide mechanism to replace DDTS. A bridge between the products based on the new Perforce Defect Tracking Gateway (P4DTG) is planned. Some highlights of the capabilities provided by Software Planner and this bridge are:

- WEB based interface.
- Support for project specific workflows.
- Support for project specific field configurations.
- Both unidirectional and bi-directional field value transfers between Software Planner and Perforce jobs.
- Support for team specific requirements, test cases, and project management.
- Graphical dashboards allowing teams to quickly identify trends and details.

Additional plans are to couple build automation with Perforce jobs. With bi-directional field value transfers this coupling provides build automation with direct communication to both Perforce and Software Planner through a single interface. This will allow users to leverage the many defect tracking and team collaboration features of Software Planner using real-time information provided by automated build systems.

Summary

The success of this project was due to many factors including senior-level management sponsorship, experienced leadership, willing participation by internal project advocates, and a clear understanding of the goals.

Bottom line: in order to successfully complete a complex transition in an ongoing development environment you must be flexible and adaptable because things will never go as planned.

For More Information

The Perforce web site contains a wealth of information that is critical for any process such as this. Some of the references specifically mentioned in this paper are:

For general information about many Perforce topics see “*Practical Perforce*” by Laura Wingerd.

For additional information about the baseline protocol see “*The flow of Change*” by Laura Wingerd, Perforce 2005 user conference.

For additional information about working disconnected see Technical Note NOTE002.

For additional information about case sensitivity see Technical Note NOTE003.

For additional information about vendor branches see Technical Note NOTE015.

For additional information about .NET and web projects see Technical Note NOTE064.

For more information, or to contact WSI:

Weather Services International
400 Minuteman Road
Andover, MA 01810
978-983-6300
www.WSI.com

For more information, or to contact SageRight:

508-393-6917
www.SageRight.com
nealf@SageRight.com