Source Control Candidates Comparison

# Recommendation

Subversion best suits our source control needs and should be adopted as our source control system of choice.

# Source Control Provider Summary

Team Foundation Server

Team Foundation Server is a suite of tightly integrated tools aimed at the enterprise marketplace as a comprehensive solution for application lifecycle management. Its strength and value are only fully recognized when the suite is adopted as a whole and used as a driver for development process and management.

As a stand-alone source control project it is a mediocre offering, clinging to many of the paradigms established by Visual Source Safe. It favors a team of local developers working on a LAN in semi-isolation using an exclusive check-out approach.

The network chatter incurred by a tight coupling between the source control server, the other TFS components, and Visual Studio makes managing source control items a heavier process. This is especially prevalent for items that are not directly part of a Visual Studio solution, such documents and SQL Scripts. As we’re not intending to use the suite to its extent, we’re paying the costs of this without reaping the benefits.

Subversion

Subversion is a stand-alone version control system designed to support distributed development teams. As a result, it has optimizations to minimize network traffic and offers solid support for the edit-merge-continue approach to development.

Subversion offers robust API support to allow it to be extended and integrated into development processes, and a good number of products exist, both commercial and open source, to provide integration with task management, automated builds, and other process tools. This will allow us to choose the proper tools to support our process and implement them incrementally.

Because it is focused on providing source code control functionality only, Subversion provides a lighter process for managing source code items without integration costs and heavy server dependencies.

# Comparison

Subversion Strengths versus Team Foundation Server Weaknesses

|  |  |
| --- | --- |
| **Subversion (Strengths)** | **Team Foundation Server (Weaknesses)** |
| * Designed to be used in a distributed environment. No persistent connection to a server is assumed. Network use is infrequent, efficient, and controlled by the client. Optimized performance for a remote team. [Can be configured as a secure service to allow for non-VPN use.] * Easily integrated into Windows Explorer, allowing for convenient and efficient management of non-Visual Studio items such as documents and SQL Scripts. * Due to the expectation of distributed development, branching, merging, and patching are considered core activities and have solid built-in support as well as support for third party tools. * Built around edit-merge-continue workflow. Support for seamless edit-merge-continue as well as exclusive file locks, when needed. Can be mixed as desired within the same project. No unneeded friction caused by requiring check-outs. * Not bound to any specific client products. Many clients are available, both commercial and open source. If desired, a developer may even use multiple clients concurrently against the same working copy. Developers need not be using the same client, allowing for freedom of personal preference. [Note: Tortoise is the most popular, and considered the de-facto standard. It would be my recommendation for a new user.] * Source control is directory-based. Managed items need not be included in a Visual Studio solution to be included in source control. Any item from the file system may be managed independently. * No server-controlled working paths or bindings. The client is responsible for tracking its associated server and relevant URLs. Top level working directories can be managed (copied, moved, deleted, etc.) as simple file system objects without any impact to subversion functionality. * Source control operations can target the subversion server repository, the working copy, or both. No working copy is needed to perform tasks such as branching and merging. [This is helpful in build automation.] * Local changes can be grouped into change sets, which can be committed as atomic units of work. * Can be integrated with active directory. * Extremely large user base with an active community. * Many add-ons, tools, and integration products exist, both commercial and open. * Extensible and scriptable via API libraries. * Open source software. Free. | * Heavily integrated and bound to its server of origin. Communication with the server is both automatic [non-configurable] and frequent. This causes a performance impact to remote users, especially those operating on VPN. [Some performance impact can be mitigated by using secure service access over a non-VPN server. Minimal support for “offline” operations.] * No way to bypass check-in / check-out model. Due to this and the heavy dependence on Visual Studio, managing non-Visual Studio assets such as SQL scripts, documents, and images within source control becomes a multi-step process. To do so, one must perform the check-out in Visual Studio, interact with the item in the preferred application, and then check-in using Visual Studio. * Heavily bound to Visual Studio. No stand alone client. Performing any source control actions requires the use of Visual Studio or the command line. All items under source control must be part of the Visual Studio solution unless the command line client is used. * Forces user-specific connection information into the solution, causing contention if users refer to the server different ways such as use of IP address vs. server alias. Result is users being forced to check-out / check-in solution file for meaningless churn. [Can be managed by convention/process.] * Adding additional resources requires additional licenses. This extends not only to developers, but also to management and business stake holders (for task management and reporting.) * Operations (such as branching) cannot be performed directly on the server. A workspace mapping and local copy is required. [Unsure of how this may impact the build.] * Command line operations will sometimes invoke GUI dialogs, unless specific option is specified. It is not always apparent which commands have implicit GUI results. This could hamper build automation, resulting in hard-to-find errors. [example, request a check-in using “tf checkin” while in a working directory with modified items.] * Unintended check-ins cannot be reverted within the GUI. The command line client must be used. |

Team Foundation Server Strengths versus Subversion Weaknesses

|  |  |
| --- | --- |
| **Team Foundation Server (Strengths)** | **Subversion (Weaknesses)** |
| * Provides a cohesive framework for enterprise application life cycle management providing development support and management visibility. [The greatest value is realized when using all components as the driving tools of an integrated process.] * Allows enforcement of process during check-in, including commenting, work item associations, static analysis, and unit test passage. * Familiar user interface for former VSS users. Most paradigms are preserved. * Tight integration with Visual Studio. [I view this as a negative, but those accustomed to VSS will be comforted by it.] * Targeted at enterprise development. Branching, merging, and shelving are considered core activities. * Local changes can be grouped into change sets, which can be committed as atomic units of work. * Integrated with Active Directory. * Granular permissioning for source control operations within a given project. * Active community with both commercial and open source add-ons, tools, and support. * Extensible and scriptable via API libraries. * Integrates well into a Microsoft server environment, and should be somewhat familiar to Windows administrators. * Supported by Microsoft. | * No built-in IDE integration. [I view this as a positive, but some used to VSS will find it initially unfamiliar.] * Source control only. Task integration, process support, and build management must be performed by other products. * Server modifications are not immediately visible unless the local client is configured to ask for them. * Moving and deleting items within a Visual Studio project is a two step process, as it needs to be performed in Subversion and the IDE. * The server has no knowledge of local edits, making it impossible to see who is editing a particular file until it is committed. * Not directly owned or supported by a single organization. However, multiple vendors offer support, training, installation, and integration. * Built around open source technologies. Some aspects not familiar to Windows-focused administrators. [Mitigated by packages such as the VisualSVN server product.] |