Addison Babcock

Submitted August 12, 2016

Source Code Management Systems

Contents

[Introduction 2](#_Toc458772759)

[Subversion 2](#_Toc458772760)

[Architecture 2](#_Toc458772761)

[Capabilities 3](#_Toc458772762)

[Typical Uses 3](#_Toc458772763)

[Team Foundation Server 4](#_Toc458772764)

[Architecture 4](#_Toc458772765)

[Capabilities 4](#_Toc458772766)

[Typical Uses 5](#_Toc458772767)

[Git 6](#_Toc458772768)

[Architecture 6](#_Toc458772769)

[Capabilities 7](#_Toc458772770)

[Typical Uses 7](#_Toc458772771)

[Conclusion 8](#_Toc458772772)

[Guidance and Recommendations 8](#_Toc458772773)

[Final Thoughts 8](#_Toc458772774)

# Introduction

Developers are currently spoiled for choice when it comes to selecting which source control system their team will use. But with so much choice how are we supposed to make an informed decision? The goal of this research paper is to lay out some of the pros and cons of some of the more popular source control systems. This paper will be exploring Subversion, Team Foundation Server and Git. For each this paper will describe the overall architecture, the unique capabilities and the typical use cases. This paper will conclude by attempting to give some guidance on which system would be best for a given team.

# Subversion

## Architecture

Subversion is loosely modeled around an older source control system, CVS. It has a centralized client server architecture that allows for offline editing.

A local copy of the repository contains both a copy of the source for normal editing and another hidden copy for comparisons. When the user updates the visible copy of the file then compares it to the unmodified file the SVN client does not need to contact the server.

The server and client software are available on every major platform. A subversion server can be run through HTTP or through a custom SVN protocol. Both are functionally equivalent for normal use. For less normal use the HTTP server can be used to view the repository contents and browse the branches and revisions.

The backend data for a Subversion repository is stored mostly as binary files, which means they are basically uneditable. However, the server only touches the files while they are being actively queried. This means that making backups of the repository is a fairly easy copy/paste. Backups can also be created and updated with the svnadmin command, which is the recommended method.

## Capabilities

Subversion offers all the capabilities you would expect from a modern source code management system. Subversion has excellent support for binary files, and some Subversion clients even offers support for merging some binaries such as .docx Word files.

Subversion offers constant time and size branching which can be useful for very large repositories. Adding a branch will not take up significant amounts of disk space.

Subversion can also be customized through the use of commit hooks on the server side. A common example of a pre-commit hook is to compile the incoming commit to make sure that the source code is valid. The compiled source code can then be run through automated tests to make sure nothing obvious is broken. If a bug is detected the commit can be rejected and the developer notified.

A post-commit hook can be used to notify all of the developers on the project that a new revision is available. Typically, this would be done by email with the svnnotify command. The svnnotify command can even add the diff to the email.

## Typical Uses

Subversion finds it’s best uses in small to medium sized teams working locally. It does not have any features beyond source control. It also does not have the ability to scale to multiple servers. The limited offline capabilities (editing only) mean that using Subversion across different regions can be limited by internet access.

Subversion does tend to find frequent use in open source circles because the source code is freely available. The lack of licensing fees also helps promote this. Subversion also finds a natural home in places where CVS was once used. Overall however Subversion use is becoming less common in favor of competitors like Git and Team Foundation Server.

# Team Foundation Server

## Architecture

Team Foundation Server uses a client/server architecture with the repository stored in a SQL database. TFS is designed to support the entirety of software development including artifacts, automated builds, bug tracking and test results. As such TFS is very scalable. TFS is broken up in to 3 tiers: client tier, application tier and data tier.

The client tier is typically the Visual Studio interface. It can also be used from the command line or sometimes from Office. Other products for interacting with the client tier include Visual Studio Team Services. VSTS is an application designed for project stakeholders who don’t need a full Visual Studio install but need to be able to interact with TFS.

The application tier is installed on one or more servers and runs on top of ASP.Net. The application tier provides web services which interface between the client tier and the data tier. The application tier also provides web services which can be used directly from the browser. This allows users to view the status builds and bugs.

The data tier stores the source code and the data for the above features. The data tier is composed of several databases running on top of SQL Server. A TFS deployment will always have at least 2 databases, one for the configuration data and one for each team project collection. If reporting is configured 4 more databases are added to support it. If SharePoint integration is configured 3 more databases are added to support that integration.

A TFS database should not be edited manually, the results are unsupported. Backing up the database through normal means is possible.

## Capabilities

TFS has the typical features you would expect from a source code management system but it is very enterprise oriented and extends far beyond only providing source control. In fact, the full capabilities of TFS can be a little overwhelming to new users.

TFS can isolate repositories (called Team Project Collections) from each other. Each team project collection gets a separate database in the data tier. The history and source code are kept separate and cannot be merged into or copied from one another. A team project collection would typically map to a team of developers and testers. A team project collection predictably contains a collection of Team Projects. A single Team Project would typically map to a Visual Studio solution file and all its dependencies although it doesn’t have to.

TFS offers fully integrated automated builds. The build process can be run on one or more or shared machines. They can be set to run at specific times, triggered by a check in or triggered manually. The builds can also be set to run tests, record the results and fail the build if needed. The results can be viewed through the web services or through Visual Studio. Builds can be deployed to any network share. The build process itself can be customized through Visual Studio.

TFS can also enable gated check ins for code review. When code reviews are enabled and a junior developer checks in code, that code will not be immediately checked in. Instead it will be saved as a shelveset for a senior developer to review. Once reviewed the code can be checked in normally. TFS can be very granular with this functionality. It can be enabled on certain branches, certain folders and even file types. TFS can also have different reviewers for different paths.

## Typical Uses

TFS is most commonly used in enterprise development environments. It has very little offline capability since so much of the functionality is dependent on a connection to the server. However, is has a full suite of integrations well beyond any other system studied in this paper.

Note that just because TFS is so heavily dependent on a network connection does not mean that IT staff need to be involved in the deployment. TFS is offered as a cloud solution known as Team Foundation Service.

Much of the functionality in TFS would need to be provided by other tools if a different source code management system is chosen. This can make TFS a simpler more cost effective solution depending on the target environment.

TFS is also the most customizable to any given environment. Just about every part of TFS can be customized and tailored to suit the needs of the organization.

TFS is designed and built for Visual Studio in a Windows environment. The server tiers only run on Windows, and the main client interface is Visual Studio which only runs on Windows. If your team does not rely on Microsoft technologies TFS would be a problem to use.

# Git

## Architecture

The key factor that separates Git from other source code management systems is the decentralized architecture. Git does not have a dedicated server. Instead each client machine maintains its own full copy of the repository.

Changes to local and remote repositories are synced through a mechanism called pushing and pulling. A developer can “pull” the latest source code, make changes locally, check them in to the local repository. Then when the developer is ready the changes can be pushed back to the central repository if the developer has sufficient permissions. If not, then the developer can make a pull request which contains the changes made. The pull request can then be reviewed before being accepted in to the central repository.

Security is maintained by a system of hashes. Each commit to the repository is identified by a hash. If the contents of the commit do not match the hash the repository is corrupted and such be cloned again.

## Capabilities

As mentioned before the main capability of Git is that each client gets a full copy of the repository. This includes the full commit history, all branches and all tags. The repository must be cloned locally before any work can be done. The developer must then check out a branch from the repository to make it available for editing. Switching branches involves checking out a different branch. Check outs are a fairly quick operation because only the files which need to change are changed.

## Typical Uses

Git was designed for large scale open source projects, in particular the Linux kernel. This isn’t to say that Git isn’t also great for smaller projects. Git finds the majority of its use in the open source development world across a wide range of project sizes. Much like Subversion, Git is both free and open source.

Git has started to find traction in enterprise environments. Developers are often familiar with Git from college which presents less training barrier for organizations looking to bring in new developers.

Some organizations might find the server-less nature of Git off-putting. The fact that the entire repository needs to be downloaded can make it quite easy for a complete copy of an organizations source code to walk out the door. This is an intentional design feature of Git because the point of open source software is sharing the source. But the ease of sharing can be a weakness for some organizations which have very secretive source code.

# Conclusion

## Guidance and Recommendations

Subversion is best used for small to medium sized projects that are ok with or need a centralized server. The need to duplicate every file in the repository twice on the local disk means that Subversion might not work well for projects that contain a large volume of binary files that rarely change. The open source nature of Subversion might also scare off some management, but the project is backed by Apache.

The major strength of Team Foundation Server is its adaptability and integration with other Microsoft products. Team Foundation Server is much more of a complete Application Lifecycle Management solution where others only provide source control. This can be overkill for organizations which already have most of their ALM solved. TFS can also be restricted by the Windows-only nature of the product. There is a good chance that this could change in the future as Microsoft builds more and more development tools for other platforms.

The final system examined in this paper is Git. Git has found a lot of use in the open source community and is becoming more common for closed source projects. The odds of continued support and development on Git is quite high since so many projects depend on it. Git could find some resistance in large enterprises for many of the same reasons Subversion might. Git provides only source control with no ability to cleanly interface with other applications. One problem that Git can encounter is the ability for the entire repository to easily be leaked. Of course it is very easy in other source control systems to obtain the latest source code. However; with Git it’s just as easy to obtain a copy of the entire repository history.

## Final Thoughts

Developers are currently quite spoiled for choice when it comes to source code management systems. There are dozens of choices and even some large organizations are building their own systems for internal use. This paper examined a small selection of the available systems: Subversion, Team Foundation Server and Git.