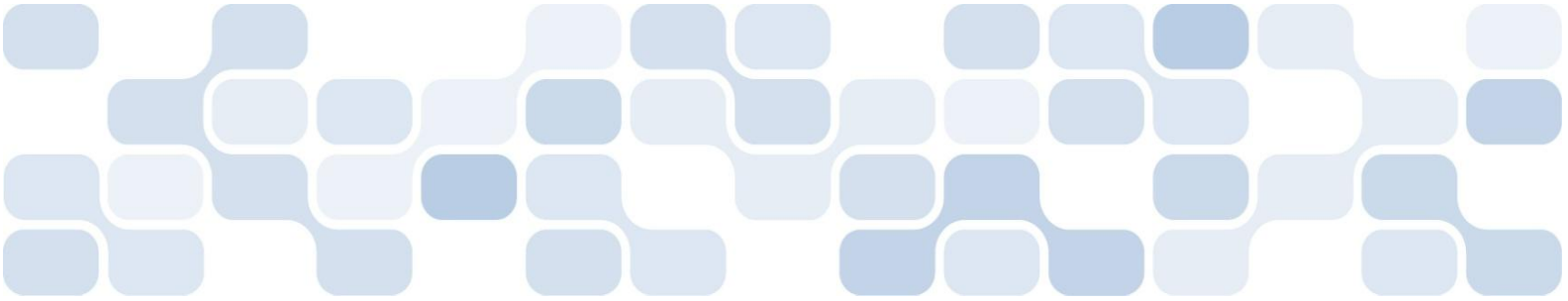
Microsoft

Integrate Work Frequently with Visual Studio Team System 2008

White Paper

May 2008

For the latest information, please see www.microsoft.com/teamsystem



CONTENTS

INTRODUCTION

Software development involves a feedback loop where code is written, tested, deployed, and ultimately its correctness is communicated back to the development team. The longer it takes for this feedback to reach the team, the more difficult it is for the team to make necessary changes and the longer it takes for new features and bug fixes to be completed. Microsoft® Visual Studio® Team System 2008 offers several ways to shorten this feedback loop, providing development teams with better information about the correctness of their code, faster than ever before.

COMPILATION, INTEGRATION, AND DEPLOYMENT

Three areas in which many software development teams may have room for improvement are compilation, integration, and deployment. Often the difficulty of accomplishing these tasks in large projects results in developer resources being dedicated to the task of ensuring the project can be built, or ensuring disparate teams' projects work correctly with one another. Often a successful build or deployment cannot take place without the oversight of these key individuals and their arcane knowledge of the steps required to pull it off successfully.

Compilation of a .NET Framework based application involves running the source code through a compiler, such as csc.exe or vbc.exe, in order to produce an output assembly. Different kinds of .NET Framework based projects require different compilation options, and the compilers have support for dozens of different settings and variables. It can be challenging to consistently compile (or build) a .NET Framework based project using the exact same process and settings time after time during a project's life cycle.

Integration of .NET Framework based applications refers to the process of ensuring different projects are compatible with one another and can work together cohesively. Very few software applications involve only a single .NET assembly, and often changes made in one assembly (or in its build process) can result in errors in dependent assemblies. These errors often do not manifest during the compilation phase, and are only discovered during integration.

Deployment of software built using the .NET Framework can involve many different variables and tasks. At a minimum, it usually involves copying a set of files and perhaps packaging them up into some kind of deployment or installation package. Additionally it can include signing of assemblies, updates to configuration files, packaging of data sources, and many other tasks specific to each project. Complex deployment processes are often prone to failure unless they are automated.

Team System addresses these issues with two important features: a robust and scalable version control system and a powerful and flexible build server. Combined, these two features help development teams to collaborate and integrate with minimal friction and provide repeatable builds of the project in an automated fashion.

COMMON SOURCE OF PAIN

Small applications built by single developers are usually fairly simple to build and deploy, and as a rule do not require any integration effort. However, the moment a second developer joins the team, a host of issues arise that can make all three of these tasks more painful, and it only gets worse as the number of developers on the team continues to grow. Some common pain points on larger development teams include (but are not limited to):

- Mismatched versions of sub-project libraries
- Incompatible configuration files between team members (or teams)
- Incompatible project or solution structures between team members
- Difficulties merging changes to files made by multiple team members
- Different folder structures and root paths and projects with dependencies on expected file system resources
- Difficulty maintaining separate versions of code for past releases or customer-specific versions
- As project dependencies grow, getting all of the files needed for a successful build becomes increasingly difficult
- Larger projects can have complex deployment requirements
- And many more

Without any version control software, many of these issues can be extremely difficult and time consuming to overcome. Further, the more difficult the tasks of integrating, compiling, and deploying the project are, the less often they will be attempted, resulting in fewer releases and a great lag between bugs being found and fixes being released.

Limited Key Individuals Possess Necessary Knowledge

Because of the difficulty that can be involved in correctly deploying a build of a large project, it is a common practice to dedicate one or more individuals on the team to the task of integrating, compiling, and deploying the project. The “Builder” must understand the file system and is often responsible for dictating to the rest of the team how the file system will be organized for the project. This individual must be able to access the required source code to compile the project, resolving integration issues in the process. Finally, the “Builder” must know how to deploy the project, usually with separate processes for dev, QA, and release deployments.

While it makes sense to have one individual dedicated to the task and responsible for knowing how to properly do it (rather than letting any developer attempt a complex build process they do not fully understand), it can be very expensive to devote one (or more) developer resources to this task. It also presents an organizational “single point of failure” – what happens when it’s time for a deployment and the “Builder” falls ill?

Each Build and Deployment is a Major Undertaking

In large projects, it is common for every build and deployment to be a major undertaking whose success is uncertain. Often the “Builder” role emerges as

a direct result of the increasing complexity of the build and deployment process and the waning confidence the team has in its ability to successfully build and deliver the project without some setback costing hours or days in the process. Even with a dedicated “Builder” role, many large projects can take hours or even days of the Builder’s time to build and deploy.

Quality Assurance Resources Idle

If producing a working application is a major undertaking, it won’t happen as often as if it were simple or automatic. When quality assurance teams do not have the latest version of the project to work with, they must either work with often-obsolete versions of the project or sit idle waiting for the next good build to test.

Difficult to Detect and Correct Regression Bugs

Regression bugs occur when enhancements or bug fixes made to one part of an application result in new bugs elsewhere in the system. Regression bugs are particularly common in tightly coupled systems that lack automated test suites, and they are among the most annoying and difficult bugs to detect and correct because they tend to occur in areas of the application that are not being actively worked on, but which depend on areas that are under construction.

VERSION CONTROL IN TEAM FOUNDATION SERVER

Microsoft Visual Studio Team System 2008 Team Foundation Server includes a full-featured version control system built on Microsoft SQL Server. Team Foundation Server's version control includes support for branching and merging, check-in policies, work item association, shelving, workspaces, changesets, and diffing/comparing. These features address many of the pain points associated with software development in team environments.

Robust and Scalable

Built on Microsoft SQL Server™, a true enterprise RDBMS, Team Foundation Server's version control system is extremely robust and scalable. Some version control systems use the file system for their data storage, and as a result often encounter problems with file locking and corruption. By using a true database for its storage, Team Foundation Server's version control avoids these issues while offering excellent reporting capabilities as well.

Branching and Merging

Branching and merging refer to two sides of the same coin. Between them, they enable developers to work in parallel on the same set of files without risk of colliding with one another while work is in progress. When a team needs to work on a common area of an application but does not wish to impede progress for other teams, it branches the code to be worked on (essentially creating a copy of the files in a separate location in source control) and performs their updates. When the updates are complete, the reverse process, merging, takes their changes and reincorporates them into the main source tree in source control.

Another common scenario for branching is to create a separate branch for separate shipping versions of the product. This enables new work to be done on the next version of the application, while bug fixes can be applied to previous versions of the application. Without branching the previous version, any bug fixes made would only be part of the latest version of the application, since new features (complete or in progress) would likely already exist in the code.

See Microsoft Team Foundation Server Branching Guidance for more information on how to use branching and merging effectively.

Check-in Policies and Work Item Association

Team Foundation Server's version control provides support for check-in policies, which require developers to follow certain steps when checking code back into source control. Some common policies include requiring the developer to provide some comments about what was changed, or requiring that all check-ins be associated with a work item. Work items describe bugs, features, tasks, and other units of work in the system, and can be directly associated with individual check-ins, making it easy during a code review to

see exactly which files were changed as part of the completion of a particular feature or bug. Team Foundation Server ships with several different check-in policies by default, and many more are available from the Team Foundation Server developer community online (or you can write your own, of course).

Shelving

Shelving enables a developer to set aside a group of pending changes temporarily in Team Foundation version control without checking them in. This enables several scenarios that improve team productivity, such as enabling a developer to set aside a long-running set of changes in order to focus on a higher priority task, or helping a developer to share a set of changes with another developer without checking them in, perhaps as part of a review process. Shelving also protects work-in-progress code from developer machine failures, and some teams require that all work-in-progress code be shelved when developers leave for the day.

A set of files to be shelved together are called a shelveset. Likewise, when checking in code, each set of files associated with a check-in is known as a changeset. Shelving code is as easy as checking it in, and as Figure 1 shows, shelvesets can include comments, associated work items, and notes just as changesets can.

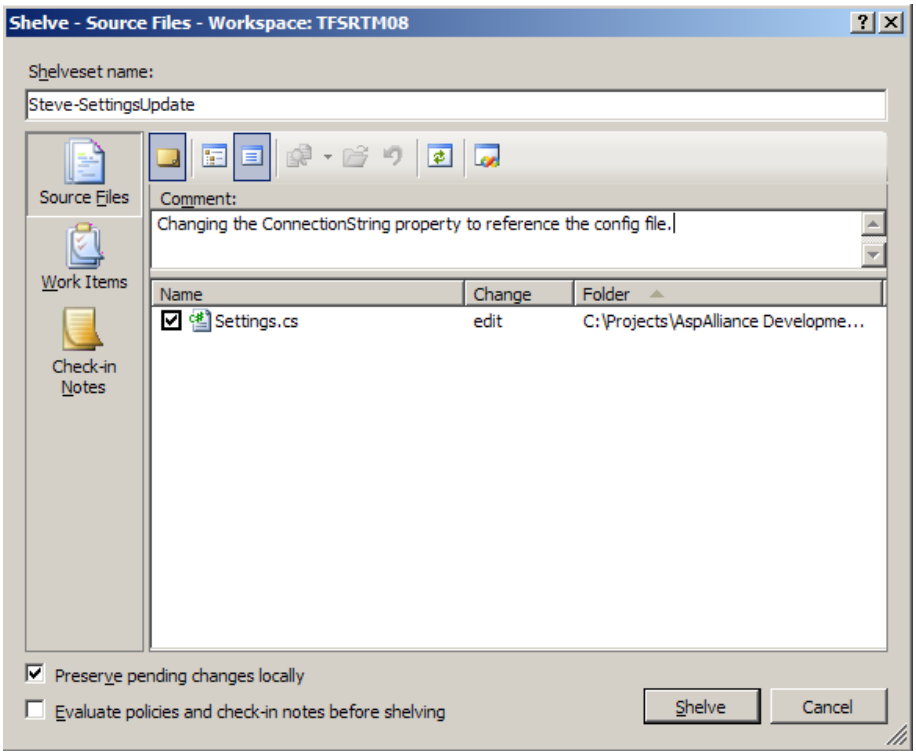


Figure 1. Shelving is similar to checking in; changes are keyed to a Shelveset name.

To learn more about Shelving in Team Foundation version control, see the following Walkthrough: Shelving Source Control Items.

Get Latest on Checkout

The new “Get latest version of item on checkout” checkbox, which you’ll find in the Options (as shown in Figure 2) dialog box in Visual Studio, will save many developers from having to merge changes on files they are working on. Without this checkbox enabled, editing a file will, by default, check out the file but will not perform a get latest. Thus, a developer who does not regularly perform a get latest before making edits to a file may find they are often working on out-of-date copies of files, and must therefore merge their changes when they perform a check-in. By enabling the “Get latest version of item on check out” a get latest will be performed when the file is checked out, ensuring that developers are always working with the latest checked-in version of a file. This simple change can save developers hours of effort.

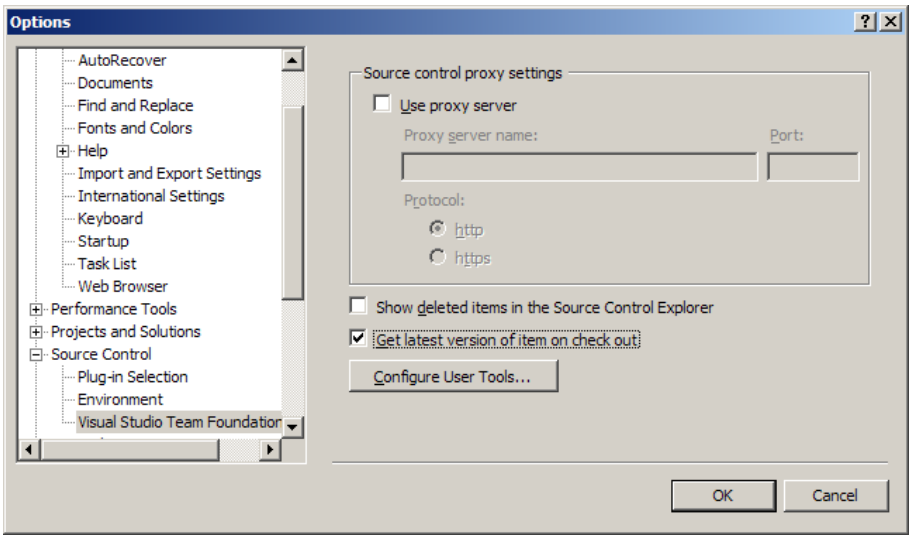


Figure 2. Enable “Get latest version of item on check out" in Visual Studio’s Options dialog box.

Key Benefits of Team Foundation Version Control

A full-featured source control repository with integrated task management and reporting enables development teams to collaborate effectively and integrate their work frequently. Branching and shelving enable parallel development, with fewer developers waiting on one another for shared files. New features like Get latest version of item on check out reduce need to merge changes and increase productivity by keeping everybody in sync with the latest version of the project’s source code.

TEAM BUILD

Team Build 2008 is a powerful build management tool that enables automated, repeatable builds (and other tasks) across multiple machines. Team Build uses MSBuild to define the tasks each build will perform, and communicates between Team Foundation Server and Build Servers using Web services. Builds are defined in Team Explorer and are then stored in Team Foundation Server's version control, making them available to the entire team and enabling versioning and change tracking.

Continuous Integration

Team Build introduces several new features which enable continuous integration, the practice of triggering a build with every committed change to a project's source files. Continuous integration provides several advantages, including rapid detection of integration errors and unit test failures, and the constant availability of a current, working build for quality assurance and release purposes. Continuous integration requires two things to work effectively:

- Builds must be automatically triggered after every check-in or group of related check-ins.
- Team members must be notified when a build fails so action can be taken to fix it as quickly as possible.

Continuous integration is a best practice for modern software development, and recommended as part of many current software development methodologies. Few projects would not be improved by the use of continuous integration, and Team Build makes the process of setting up a build server easy.

Microsoft designed two new features in Team Build to enable continuous integration scenarios. Builds now support the concept of triggers, or events which cause the build to automatically occur. By default, builds have no triggers, but you can modify this to trigger the build with every check-in or on a scheduled basis on certain days of the week (for a nightly/daily/weekly build). You can optimize this further by limiting the number of builds per check-in so that check-ins that occur while a build is in progress are batched together into one new build, or limiting builds to only occur every N minutes or more. Figure 3 shows the options for managing triggers in the Build Definition dialog box.

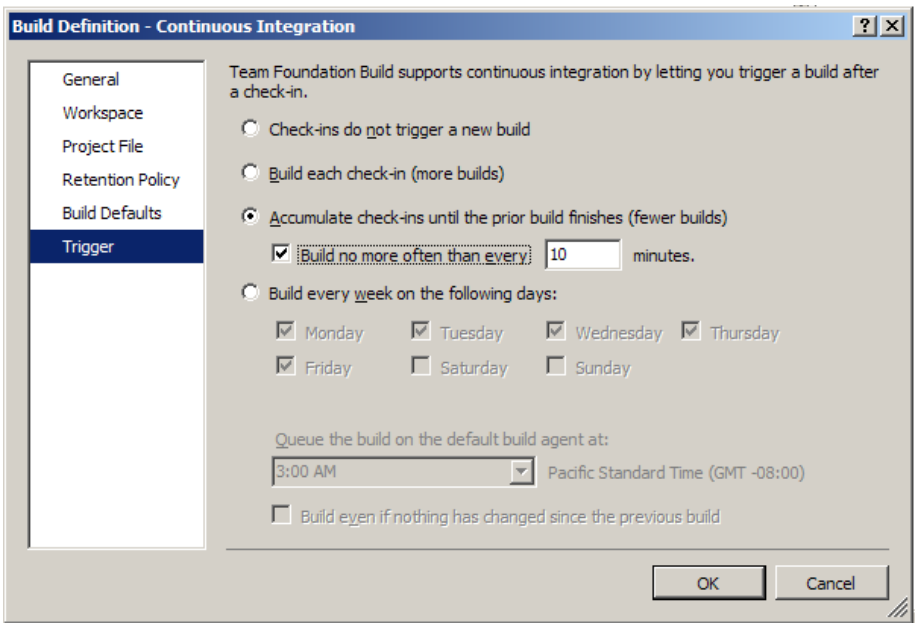


Figure 3. Triggers in Team Build 2008 allow for continuous integration.

Because continuous integration tends to lead to a much larger number of builds, Microsoft also added a feature to help manage the artifacts generated by each build. Figure 4 shows the Retention Policy menu of the Build Definition, which is used to specify how many copies of the build should be kept based on the outcome of the build.

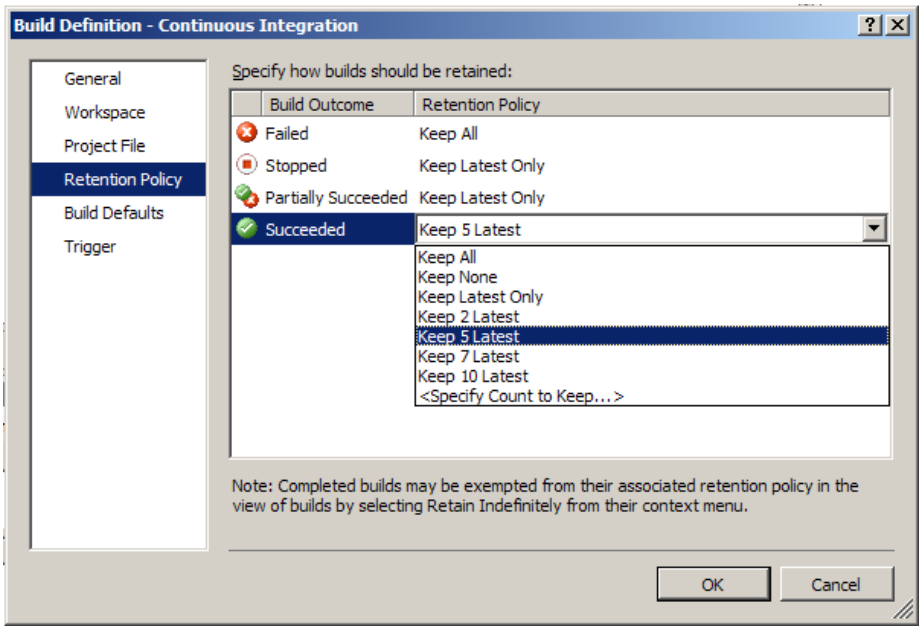


Figure 4. Retention policies limit the number of builds stored on the server.

Notifications and Reports

To close the loop with continuous integration, you need a way to notify the team when the build fails or is fixed. The Team Foundation Server Power Tools – December 2007 Release includes a Build Notification Tool that runs on users' desktops as a system tray icon. You can configure it to show notifications for builds started, queued, and/or finished for individual build definitions within team projects. Figure 5 shows the configuration options for the notification tool, and Figure 6 shows how the notifications appear in the system tray area of a user's desktop.

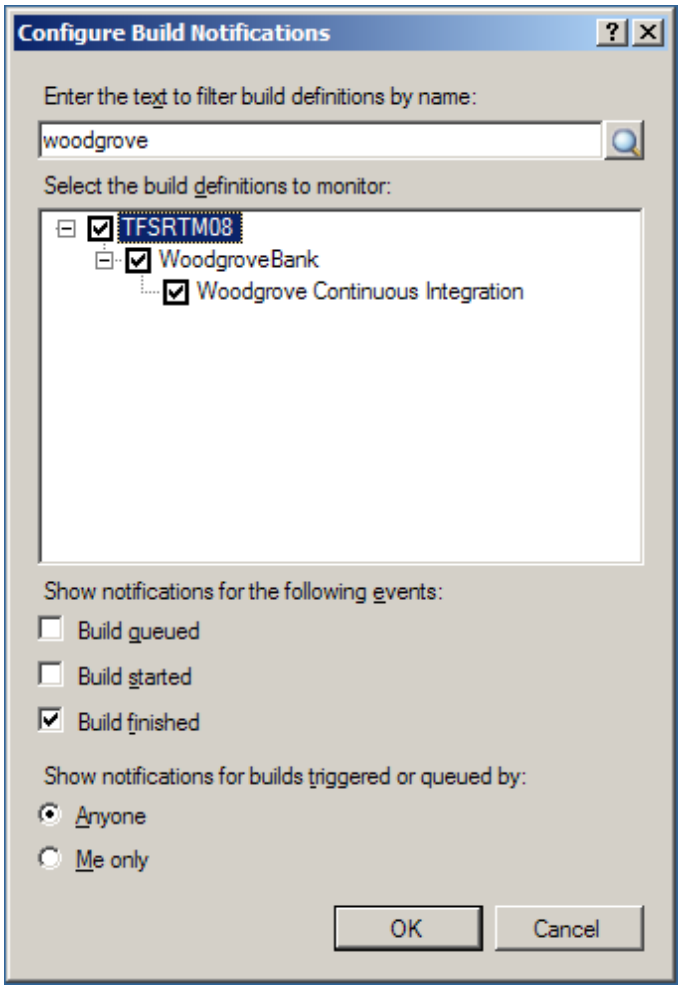


Figure 5. Configure notifications by choosing specific build definition, events, and triggering users.

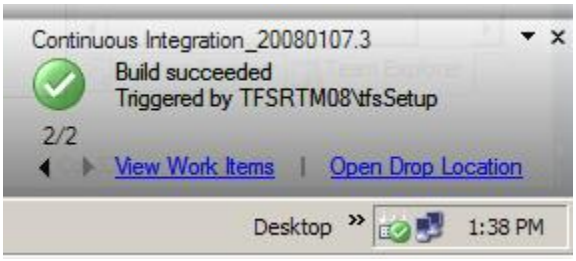


Figure 6. Build notifications appear above the system tray whenever monitored events occur.

Once a problem is introduced into the project, everyone on the development team is notified within moments. The developer responsible for the problem (typically the last one to check something in) should immediately address the issue and check in additional updates that will allow the build server to successfully build the project. This minimizes the amount of time that the automated build is broken, and ensures that bugs introduced are found and fixed immediately. It is also a good idea for teams to strive never to break the build on the build server, which can often be avoided by running tests locally before checking in changes.

In addition to notifications, Team System includes rich reporting built on Microsoft SQL Reporting Services. Interested parties can quickly view information about recent builds and check-ins, or subscribe to periodic reports covering information of interest to them. SQL Reporting Services powers the reporting engine piece of Visual Studio Team System 2008. A simple report listing builds with related build quality and test pass rate is just one of nearly twenty reports available by default from a new project. Figure 7 shows some of the sample reports available from within Team Explorer for a given team project.

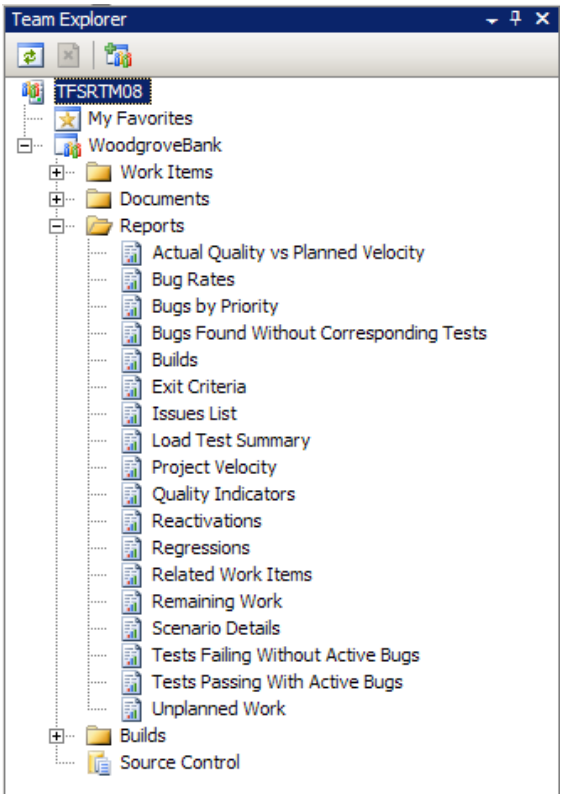


Figure 7. Team Projects include many built-in reports, and additional ones can be added as required.

Unit Tests and Code Analysis

Improving code quality is one of the key benefits of having automated, repeatable builds. Two ways this can be achieved are the use of unit tests that ensure code is doing what it is supposed to, and static code analysis that ensures code is written according to specified conventions and best practices. Using Team Build, developers can easily perform both of these processes as part of a build definition so that they occur with each build. Combined with continuous integration, this can greatly enhance code quality by ensuring that failing unit tests are discovered immediately after code is checked in.

Build Management

With Microsoft Team System, each variation of building a project is a separate build definition. You create build definitions via Team Explorer and store them in Team Foundation Server's version control. Once created, any user with the necessary privileges can access these builds from any machine, enabling any member of the project team to initiate a build. The results of each build are also recorded and included in various reports. Further, you can run builds on multiple machines, enabling a single Team Foundation Server to initiate builds on a number of servers running Team Build at the same time.

Custom Build Tasks

Because Team System build definitions rely on MSBuild project files to define the tasks they will perform, it is very easy to define custom build tasks. Such tasks may not be related to the compilation, testing, and deployment of code, necessarily, but can include anything for which an MSBuild task can be written, which is pretty open. For example, you might consider building custom build tasks to generate documentation from XML comments, schedule long-running tasks, or perform live tests of a Web site's uptime or a database's integrity. There's virtually no limit to what you can define as a build definition, and with triggers and scheduling, Team System can manage custom tasks as easily as it can manage a project's build process.

With custom build tasks, Team Build can be used as an extremely powerful application server, running scheduled or triggered applications and workflows in response to a variety of schedules or events.

Key Benefits of Team Build

A managed build process and frequent builds boost productivity in a team by reducing the time between introduction of bugs and their discovery. Critical information about the build and deployment process is protected in source control and available to all members of the project team, eliminating the need for dedicated build personnel. Notification and reporting tools ensure information flows to project members who need it as quickly as possible.

CONCLUSION

The combination of a robust and scalable version control system with a flexible and automated build system improves productivity and quality in software development teams. By enabling developers to work together easily while at the same time quickly detecting and communicating integration issues, progress on the application can move forward at a much faster pace. By reducing the feedback loop between integration of bugs and their detection and correction, Visual Studio Team System 2008 Team Foundation Server greatly improves the efficiency of software development teams.

Additional Resources

Microsoft Team Foundation Server Branching Guidance
Team Foundation Server Developer Center
Team Foundation Server 2008 Power Tools
What's New in Team Foundation Server 2008?
Introduction to Team Build 2008 for Team Build 2005 Users
Team Foundation Server 2008: A basic guide to Team Build 2008

ABOUT THE AUTHOR

Steven A. Smith is a Microsoft Regional Director, ASP.NET MVP, and author of several books on ASP.NET. He runs ASPAlliance.com, a software development resource, and is CIO of Lake Quincy Media, LLC, which operates the largest online .NET developer advertising network. You can reach him via his blog: http://aspadvice.com/blogs/ssmith/.

This white paper was developed in partnership with A23 Consulting.