

Ensure Quality Early and Often with Visual Studio Team System 2008

White Paper

May 2008

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



This is a preliminary document and may be changed substantially prior to final commercial release of the software described herein.

The information contained in this document represents the current view of Microsoft Corporation on the issues discussed as of the date of publication. Because Microsoft must respond to changing market conditions, it should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information presented after the date of publication.

This white paper is for informational purposes only. MICROSOFT MAKES NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, AS TO THE INFORMATION IN THIS DOCUMENT.

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

© 2008 Microsoft Corporation. All rights reserved.

Microsoft, Excel, Outlook, Visio, and Visual Studio are trademarks of the Microsoft group of companies

All other trademarks are property of their respective owners.

CONTENTS

Introduction	1
Quality Before Coding	2
Quality During Coding	5
Quality after Coding	10
Conclusion.....	12
About the Author.....	13

INTRODUCTION

Developing quality applications is critical to business success as businesses depend on these applications when deployed in production. One of the biggest challenges for developers today is that code that is well designed, implemented, and unit tested often fails in production mission critical environments. This is supported by the fact that 74% of the problems found in production environments are found by end users. How then can developers and testers ensure that software quality is maintained and delivered across the application life cycle?

Microsoft® Visual Studio® Team System 2008 introduces new features and builds on existing ones to enhance the development experience across the application life cycle. One key area of software development has always been quality. In the past this meant performing extensive tests on a working version of the code, or at least working subsets of the code. While such functional tests may still be performed, it's better to start ensuring quality earlier in the process. Visual Studio Team System 2008 contains a wealth of features that help ensure quality before coding begins, during the coding process itself, and after coding is finished.

Ensuring quality at various stages makes sense both in terms of schedules and budgets. Analysts and architects verify the design and that it will perform on the current architecture. Developers test code while it is being written, see how much code is actually executed by tests, and perform analysis checking for known performance or security issues. Testers automate Web and load tests and check the code throughout the process. Any problems found can be flagged as bugs and assigned to people through the Team Foundation Server's work item tracking.

QUALITY BEFORE CODING

When most people think of ensuring quality in a software development application, they think of testing the application at various stages of completeness by testing specific tasks; in other words, functional testing. More recently, there has been a rise in testing during the development phase using unit testing. However, there are quality tasks that can be performed before coding begins, and these tasks are supported by Visual Studio Team System 2008.

Scenarios and Requirements

Requirements Gathering

Visual Studio Team System 2008 includes templates for capturing scenarios and requirements, in both Word and Excel formats. Companies are free to modify the existing templates or upload their own. Documents that have already been created can also be uploaded to become part of the project. Documenting the requirements and user scenarios up front helps document the features the application must implement in order to be successful. Storing the documents as part of the project helps ensure that the documents are easy to find and requirements are always at hand. Figure 1 shows the Requirements folder in the Team Explorer and the Scenario Description document template provided by Team System.

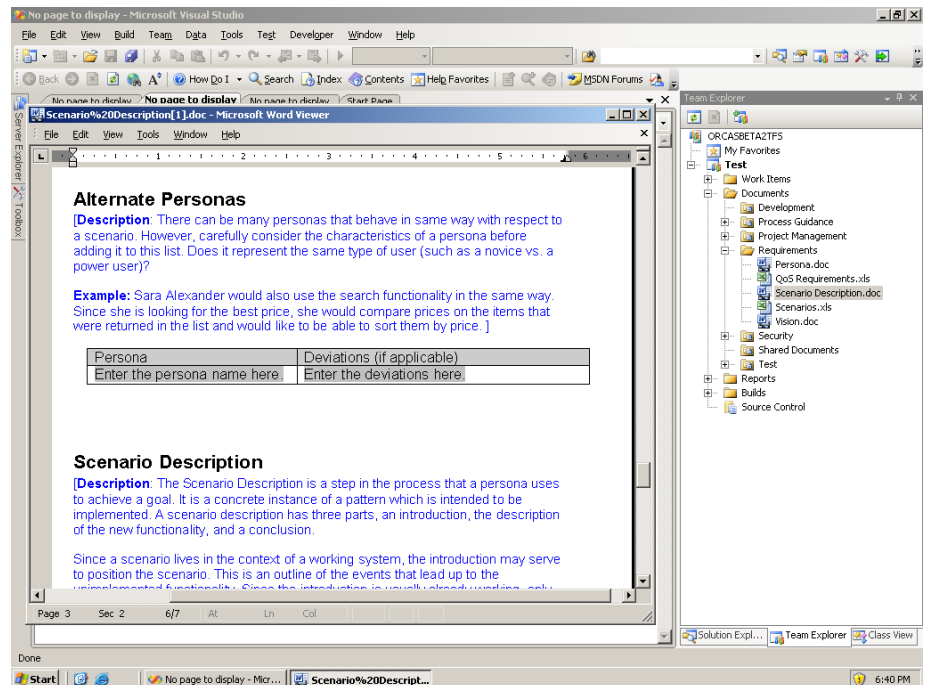


Figure 1. Visual Studio Team System 2008 provides templates for requirements gathering tasks.

Mapping Scenarios to Work Items

After scenarios and requirements are documented, those scenarios and requirements can be created as work items in Team System. Work items enable managers to track the progress of a project and for architects, developers, and testers, to see what tasks remain. The work items should encompass the requirements and scenarios documented earlier. Depending on the process template used, work items can be categorized as a Bug, Quality of Service Requirement, Risk, Scenario, Task, or a type of your own definition. When it comes to looking at quality, the most common work items are bugs, quality of service requirements, and scenarios. Bugs usually come later in the process, falling out of testing or actual customer usage. Quality of service requirements can encompass such items as the speed at which certain items must complete. Scenarios might include use cases in which a customer walks through a particular task, such as adding a new order.

Assigning Work Items to Team Members

Once work items have been created, they can be assigned to different team members. These team members may include application architects, infrastructure architects, developers, database professionals, and testers. Each person on a project is able to see all tasks assigned to them at any time, and tie any work they do to one or more tasks. This provides a consistent visibility into the project across various teams ensuring project success. In this way, the team is focused on resolving the issues that have the greatest impact on the system, enforcing quality throughout the life cycle.

Verifying the Architecture

One of the great features of Visual Studio Team System 2008 Architecture Edition (and, of course, in Team Suite) is that the application architecture can be defined up front using diagrams that show how the components will be separated and how they will communicate. This enables application architects to show what services the application will use, what the front-ends look like, and how the front-ends, services, and database communicate.

The infrastructure architect can likewise lay out the infrastructure of the data center, including what services run on different servers and how the services communicate. The servers can include the definitions of the operating system version, service pack levels, and so forth.

Finally, the application architecture can be placed onto the infrastructure architecture to ensure that the application, as designed, will work on the infrastructure that exists. This can identify potential issues early, such as application dependencies on service pack levels or .NET Framework versions that are not yet installed on the servers. Figure 2 shows an example of an application mapped onto the logical infrastructure architecture.

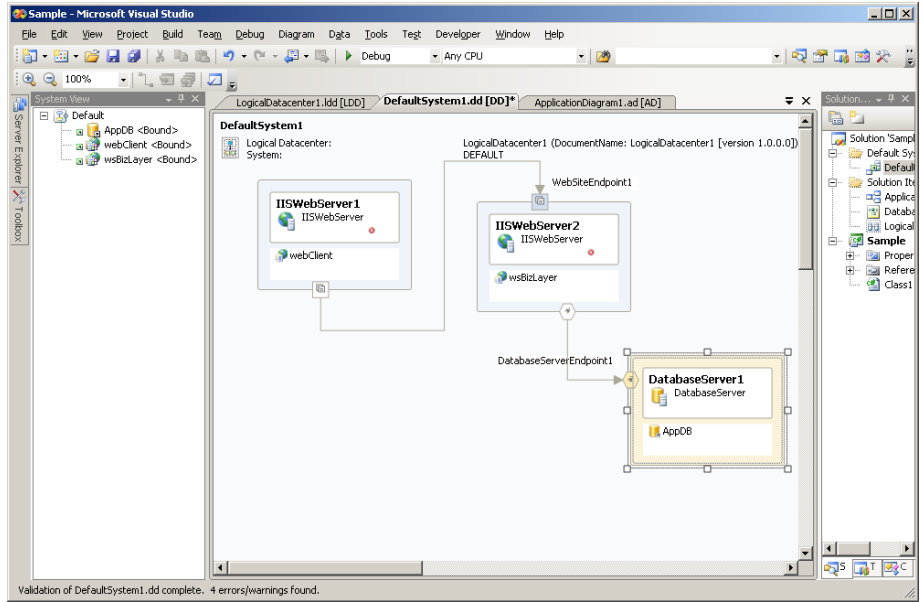


Figure 2. Mapping an application architecture to the infrastructure ensures that all the pieces are in place to support the proposed system.

QUALITY DURING CODING

In the previous section, you saw how Visual Studio Team System 2008 helps project managers, architects, developers, and testers assure that quality was in place before coding began. Work items were set up to ensure that quality of service objectives would be tracked, and the application architecture was designed and then placed on the infrastructure architecture to ensure that it would deploy and run on the hardware that was in place.

The next step is to work with the source code, and here Visual Studio Team System 2008 includes a wealth of tools aimed at supporting and enforcing the quality of the code. These features include unit testing, static code analysis, performance profiling, and more.

Source Code Control

Working with source code almost always means a need for source code control. Even small projects can benefit from the code archiving, differencing, and searching capabilities provided by the source code control found in Visual Studio Team System 2008. A great feature in Visual Studio Team System 2008 is the ability to access the version control system over the Web, using a separate add-in.

Branching and Merging

One of the techniques that can be of great help in developing systems is code branching. Before adding a new feature, the code is branched, meaning a copy is made. The new feature is implemented in a separate branch, and work on the main “trunk” can continue without being affected by the development of the new feature. Periodically the code in the branch can be refreshed to synchronize it with any changes to the main trunk of code. After the new feature is coded and fully tested, the new branch can be merged back into the main trunk. In this way, new features are not added to the main product until they have been tested; this has the added benefit that features that fail to make the grade can be abandoned without having to remove any code from the main trunk.

Restoring Previous Versions of Code

As helpful as branching and merging may be, there are times when new code is compiled and tested and the decision is made to roll back to the previous version. Source code control doesn’t simply roll back as it used to, as this doesn’t enable proper tracking under laws such as Sarbanes-Oxley. Instead, Visual Studio Team System 2008 gives the appearance of rolling back by taking the chosen previous version and making it the current version with a new version number. This helps ensure that the system is not taken offline due to updates that cause issues.

Code Analysis

Static Code Analysis is a process by which code is examined in more detail than by the compiler—the code is actually checked against a set of rules.

The results of the checks can raise warnings and errors. This is similar to a structured code walkthrough except it is performed programmatically rather than by a room full of people.

Visual Studio Team System 2008 enhances code analysis by adding static analysis to T-SQL code and checking for quality and security issues. Code analysis rules for other languages have been enhanced to increase the accuracy of the suggestions.

Developers can perform code analysis on an ad hoc basis or a project manager can require it as part of the check in process through a check-in policy in Team Foundation Server. This ensures that someone has examined the code and has addressed any errors or warnings. Figure 3 shows that even simple code can generate warnings, and that you can turn specific checks on and off as needed.

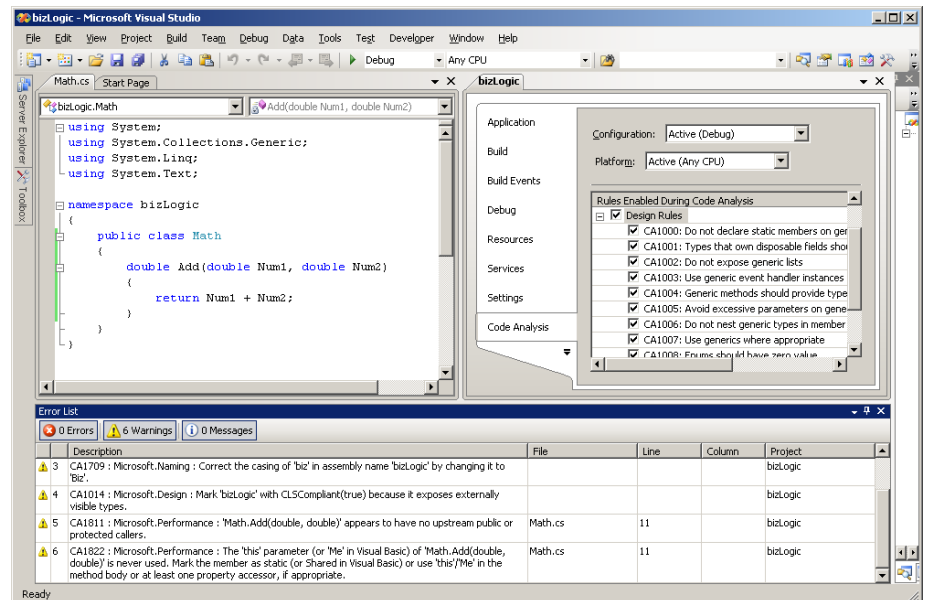


Figure 3. A simple C# class generates six warnings (bottom) while the rules that are checked can be turned on and off (upper right).

Performance Profiling

Businesses always want applications to run as quickly as possible, and developers often compete amongst themselves to have code that runs as efficiently as possible. Code that runs slowly can be a challenge to debug, and one tool to help identify problem areas is the Visual Studio Team System Profiler.

Identifying Performance Bottlenecks with Hot Path

Using the Performance Profiler in Team System, developers can create performance reports that show the time spent in each part of the application. Visual Studio 2005 Team System supported this feature but finding the slowest areas of the application often meant opening a call tree view and

following the slowest path, expanding until you located the bottleneck. This could take a tremendous amount of time as the call tree could be deeply nested (meaning the offending method or framework call might be twenty levels deep). Hot Path analysis, new to Visual Studio Team System 2008, adds a toolbar button that automatically expands the call tree to identify areas that have high inclusion times compared to siblings.

Collecting Windows Counters

Visual Studio Team System 2008 features profiling which developers can use to collect Windows counter information while an application runs. During a performance session, developers can choose to collect various performance counters, such as memory and CPU usage. After running the application, a developer can examine performance reports to show the values of those metrics during runtime. Figure 4 shows how to collect various counters as part of profiling an application.

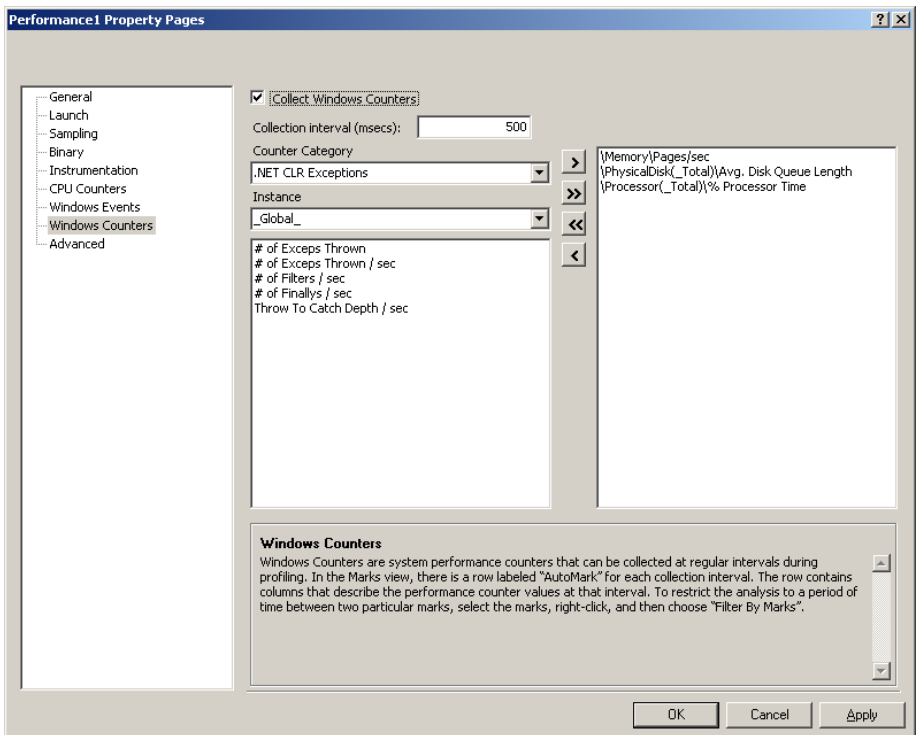


Figure 4. You can collect Windows counters during the profiling of an application, showing such items as the memory and CPU usage as the application runs.

Unit Testing

When most people think of ensuring quality for applications, they think of functional testing; that is, testing the application after it is complete to one degree or another. However, there has been a trend over the past 7-10 years to test the code early and often during the development phase using unit testing. This is done using the same language as the code rather than

learning a new testing language, and it is set up in such a way that tests can be run at nearly any time to verify that the code, as written, passes the tests that have been created. At first, this helps ensure that the code is functioning properly, assuming the tests cover the required functionality of the code. Later, the unit tests help support changes because developers can always rerun existing tests to make sure they have not broken functionality that used to work.

Test-Driven Development

Test-Driven Development (TDD), also known as Test-First Development, is a development paradigm that says a developer writes a test before writing any code. After they've written the test, the developer writes just enough code to get the test to pass. Once the test passes, the developer writes the next test and writes enough code to get that test to pass, making sure that no previous tests break. Through this process, a suite of unit tests are created and the code is verified as passing all tests.

First class support for unit tests can be found in Visual Studio Team System 2008 as well as the Visual Studio Professional 2008 products. Through the creation of test projects, users can add unit tests to a solution and then run them at any time, showing success or failure through a simple green light/red light metaphor. Of course, unit tests can also be run as part of an automated build process.

Real-world Quality from TDD

Unit testing can certainly be done without implementing TDD, but TDD has been proven effective in university studies. Dr. Laurie Williams, a professor at North Carolina State University, performed a study that found that defects identified by customers were 30% lower for a system using TDD than for a previous version of the system that did not . Another study found that 95.8% of developers felt that TDD reduced the debugging effort .

Code Coverage

When creating unit tests, the goal is to write a test and then write the code that causes the test to pass. In reality, developers often write more code than is required and therefore might write code for tests which do not exist. A common example is the case of an If statement in which one branch is never tested. Team System includes a code coverage tool that shows the percentage of code covered by tests, the percentage not covered, and more importantly, color codes the source code to easily identify lines that were and lines that were not touched by unit tests. Figure 5 shows the code coverage results for two simple methods in a Visual Basic class.

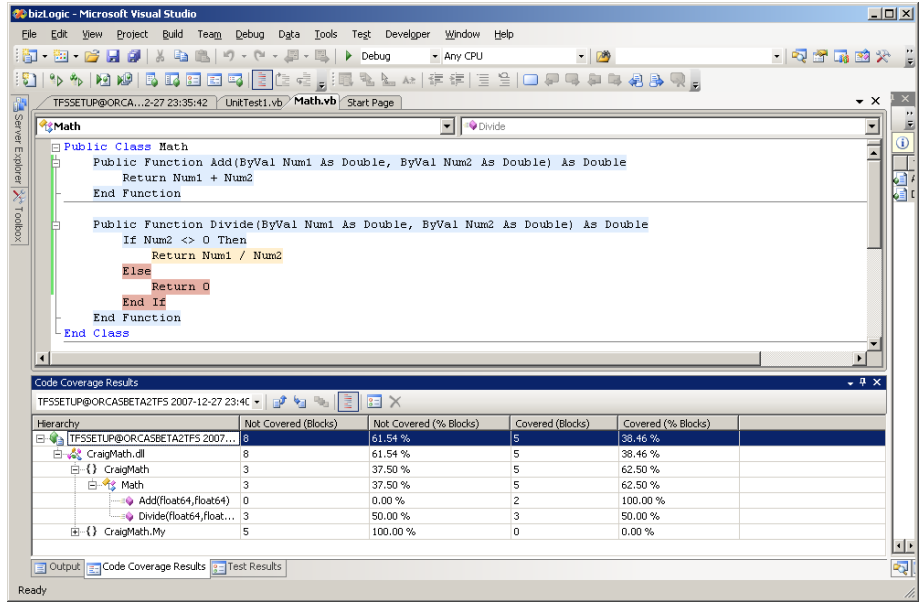


Figure 5. Code coverage results shows that Else portion of the If statement is not touched by any of the unit tests that were just run.

Database Unit Testing

While unit testing is normally considered a tool for source code in programming languages such as C# and Visual Basic, there are applications for unit testing with T-SQL in areas such as triggers and stored procedures. Visual Studio Team System 2008 Database Edition includes tools to create and run unit tests for stored procedures and triggers. The unit test, as well as other database artifacts – including database schemas – in source code control. This enables developers to treat databases as first-class citizens with the source code, ensuring that changes to stored procedures and triggers can be tested and then deployed only if all tests pass.

Reporting Progress to Stakeholders

Project stakeholders, be they inside the IT organization or in other areas of the business, are able to access a project portal that is created as part of the Team System project. This portal is hosted in SharePoint and contains a series of preconfigured SQL Server Reporting Services reports. By using these reports, stakeholders can monitor the progress of the application through bug tracking statistics, testing success rates, and more. These reports are always up to date because they access the Team System database in real time. This provides an excellent resource for customers throughout the organization to keep apprised of the project's progress.

QUALITY AFTER CODING

So far, quality checks have been done before any coding started, and then throughout the coding process. Naturally, testing can be continued when the application is turned over to a testing team. Fortunately, Visual Studio Team System and Team Foundation Server can automate the building and testing of applications to a large degree.

Automated Builds and Testing

Visual Studio Team System includes an automated build engine called Team Build. This build engine enables a variety of build definitions to support regularly scheduled builds, release builds and an agile technique known as continuous integration builds—a process in which builds occur either immediately after a check-in, or in short intervals, queuing a small number of check-ins before a build. A Team Build definition can be created to enable one to check out the source code, compile it, run static code analysis and unit tests, deploy the code to a test server, run Web and load tests, and publish results of the build and associated tests.

Web and Load Tests

There are a couple of different kinds of tests that can occur after the application has been created. These include Web tests, which may or may not be used to perform load tests.

Web Tests

Web tests are a way to test the functionality of a Web application. Developers often record such a test when a user walks through the process of using a Web application, which records the HTTP Requests and Responses. In addition, Web tests can support ASP.NET AJAX, enabling for testing of rich Web applications that include client-side code. Team System can do more than simply record a Web test—it can enable developers to pull selections from a database, XML file, or CSV file. For example, imagine that a user drops down a combo box and selects an item. The test can be configured to pull the items from a database or file and randomly select one for each test, providing a more realistic test suite by simulating the selections of different values rather than always choosing just the one that the user picked during the recording.

Load Tests

Load testing in Team System is powerful because it enables for Web tests, manual tests, generic tests, or any combination of such tests to be automated. Team System 2008 introduces a new User Pace test mix, which enables developers to control the mix between different test types and the number of simulated users can increase over time. For example, one test may test the order entry system while another may test the account update feature. Each user could be set to run ten order entry tests but just one account update test every hour. Load tests are useful for finding bottlenecks

not just in the application (recall the ability to add performance counters to code) but also hardware bottlenecks. Different browsers can be simulated with Web tests and performance counters from target machines can be captured for later analysis of the load resulting from different browsers and different access methods. Different network bandwidth types can be simulated to cover local and remote access scenarios and check for both performance and load issues.

Once the load testing has started, performance can be monitored in real time. After the tests are over, statistics are captured for review and analysis.

CONCLUSION

Visual Studio Team System 2008 provides a series of powerful, integrated tools that assist organizations with ensuring the highest quality in their software development throughout the application life cycle. From up front planning through coding to delivery, people at every phase verify, test, profile, monitor, and track quality. Even those outside the IT organization can see the progress through the project portal that is automatically updated as the project progresses.

Architects design the physical application and apply it to the existing architecture (or design a new architecture to support the application.) The design can be verified before a single line of code is written. Developers create unit tests to not only ensure that their code works, but to verify that future work does not break existing functionality. Developers can also see what parts of their code are not covered by unit tests, quickly find performance bottlenecks, and obtain suggestions for improving their code. Testers create and run automated tests to check for performance and functionality issues, monitoring a variety of counters on remote servers and in the application. Anyone in the process or even outside of IT can monitor the progress by tracking bugs, test results, and more, using the Team Foundation Server portal.

Visual Studio Team System 2008 breaks new ground in integrated support for built-in quality features. The tools and features provide an end-to-end solution for verifying and testing the application, and supporting the development team at each step by providing timely and actionable feedback.

ABOUT THE AUTHOR

Craig Utley (craig@solidq.com) is a mentor with Solid Quality Mentors and a former Program Manager on the SQL Server Customer Advisory Team at Microsoft. He splits his time between providing business intelligence solutions and consulting on developer productivity issues, including design patterns, test-driven development, and agile methodologies.

This white paper was developed in partnership with A23 Consulting.