Microsoft

Microsoft® Visual Studio®

Team System

Making Real-Time Decisions with Visual Studio Team System 2008

White Paper

May 2008

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

CONTENTS

OVERVIEW

The “center” of the Microsoft® Visual Studio® Team System 2008 Team Foundation Server universe is a Microsoft SQL Server™ OLAP Data Warehouse that tracks and stores information based on many developer activities in a Team System environment. You create a Team Project in Team Foundation Server, and crate a build definition–that data is recorded. You write code and execute unit tests–that data is recorded. You find and log bugs for your project– that data is recorded. You add, change, and remove lines of code (do you see a pattern emerging here?) that data is recorded. For nearly every activity that a team engages in there are records created in Team Foundation Server. That data tracking activity - often revealed through reports - becomes information that Project Managers, Development Leads, and other stakeholders can use to make project decisions.

The world of software development has many nouns and verbs - bugs, hours, priority, triage, test, scenario, code change, etc. - along with many other factors and activities. Getting a handle (now, not later) on what areas of your project are solid (and aren’t solid), what’s currently happening, and also possessing facts on what has happened in the past, can all be very helpful in planning as well as making on-demand decisions. Team Foundation Server helps you to turn all of this raw data from your team’s daily activities into information that you can analyze to gauge your team’s performance with respect to your goals.

Team Foundation Server provides an entire reporting infrastructure to help development teams make decisions on iteration planning, test coverage, bug trends, project schedules, resource allocation, and more. The reporting infrastructure is comprised of pre-built SQL Server 2005 Reporting Services reports, as well as an open-architecture OLAP Data Warehouse for building your own custom analysis. Team Foundation Server does this by storing all of the project information and activities and providing over a dozen reports at the click of a button.

Do you want to track bugs by priority? Are you concerned whether your QA team is finding bugs quickly enough? Do you want to verify that each build contains sufficient code coverage testing when large amounts of code are added?

This white paper will cover the following:

- The types of challenges that software project managers face
- The information items that Team Foundation Server tracks
- The Team Foundation Server reports. As an example of what you can do with the kind of data Team Foundation Server helps you gather, the paper will drill down into one specific report (the Quality Indicators report).
- How to modify/customize reports to your needs using SQL Server 2005 Reporting Services and MDX, and even how to create new reports

MANAGING A DEVELOPMENT PROJECT

A development manager must wear many hats and take on many roles. In one of those roles, a development manager must provide very current information to upper-level executives and project stakeholders about the status of a project. The manager must be able to answer questions about current and past productivity, how resources are currently being used, as well as whether additional resources may be needed in the future.

The manager also needs to manage more detailed information "in the trenches" with project team members on daily builds, prioritization of bugs, regression testing, bug rates, and utilization of each resource.

Managers also need to monitor composite statistics, such as code churn (number of lines added, modified, or deleted from a file from one version to another): they use this to measure the quantitative impact of code changes. Another common statistic that managers must consider is code coverage (the degree to which all aspects of a specific code module have been tested). Sometimes a picture *can* be worth a thousand words. For example, a sudden spike in code churn indicates your team is adding or changing more lines of code, whereas a decreased trend likely means that fewer items are being added or changed, and therefore things may be moving towards stability.

Team Foundation Server helps a development manager identify qualitative and quantitative information about key performance indicators in their team's development process, their product, or even their entire business. Reporting features in Team Foundation Server makes this information available to help managers identify where their process needs to make adjustments to make their business process better Team Foundation Server captures the type of qualitative and quantitative information to assist you in making the types of decisions necessary to support your process, your product, or even your entire business.

Key Performance Indicators to Analyze

In trying to build a management evaluation process, one of the initial struggles can be identifying not only what data is critical for the process, but also how to easily obtain it, and how to output it in a meaningful way. Team Foundation Server has business intelligence features that enable you to analyze key performance indicators about a development project, and a flexible reporting architecture to turn tracked indicators into something that everyone can easily communicate.

Any reporting architecture is only as effective as the quantity and quality of the raw underlying data used to drive the reports. Team Foundation Server tracks many key pieces of information that translate to indicators that a manager can use to not only track performance of your development team, but report progress and status and overall trends to upper management.

Team Foundation Server saves you the time of identifying and searching for key measures by providing reports that focus on important indicators such as the following:

- Code churn (lines added/changed/removed from code for a corresponding check-in/build)
- % of Code coverage (what percentage of code has been tested through unit tests). Note that you must enable code coverage for unit tests.
- Bugs unassigned, priority of bugs, etc.
- Planned work
- Unit Tests failed/passed/inconclusive
- Regression test

As you'll learn as you read on, Team Foundation Server reports show these measures in a variety of detail and summarized ways, providing thorough listings and visual representations of the data.

The Team Foundation Server Reporting Architecture

Understanding the Team Foundation Server Reporting Architecture means understanding two areas: first, the content of the reports themselves, and second, the underlying data warehouse (TfsWareHouse) used to generate the reports. This next section will list each report with a brief description, and will also cover the elements of the Team Foundation Server data warehouse that serves as a data source for the reports. After covering the general list of reports and the general structure of the TfsWarehouse, this paper will take one of the most popular reports (Quality Indicators) and talk in detail about how data "makes it" onto the report, and how a manager might effectively use the report.

Available Reports and What They Do

Table 1 describes all of the reports in Team Foundation Server, along with a brief description. Note that the Scenario Details report is only available in the MSF for Agile Software Development template, and the following four reports are only available in the MSF for CMMI Process Improvement template: Issues and Blocked Work Items, Requirement Details, Requirements Test History and Overview, and Triage.

- **Actual Quality vs. Planned Velocity**
  Bubble chart of bugs found per scenario resolved. This can provide a quick glance of how healthy a project is.
- **Bug Rates**
  Line chart of bugs active, bugs new and reactivated, and bugs resolved, by date
- **Bugs by Priority**
  Stacked column chart of bugs found and fixed by priority, by date
- **Bugs Found Without Corresponding Tests**

A listing of bugs encountered without actual tests

- **Builds**
  A listing of program builds; % of tests passed, and code churn/coverage statistics
- **Exit Criteria**
  A listing to view the status of all identified exit criteria (events that must occur before the particular process is complete)
- **Issues and Blocked Work Items**
  Report that shows remaining open issues as well as the trend toward resolving them.
- **Issues List**
  Listing of open issues, assigned to person, priority, remaining hours
- **Load Test Detail/Summary**
  Summary and detail reports on results of load testing on application performance
- **Project Velocity**
  A line chart that shows the rate of planned work resolved and closed by date
- **Quality Indicators**
  A detailed stacked column chart that is used to display overall project health: plots tests passed/failed/inconclusive as vertical bars, and code churn/code coverage/active bugs as lines.
- **Reactivations**
  A column chart showing reactivated work items versus total work items
- **Regressions**
  A listing of regression tests (tests that once passed and are now failing)
- **Related Work Items**
  A listing of work items that are related to/dependant on other work items
- **Remaining Work**
  A stacked area chart that shows active work remaining resolved and closed over time. Helps to predict when you will be at Code Complete.
- **Requirement Details/Test**
  Report shows testing results against defined scenarios and requirements.
- **Requirements Test History**
  Shows the progress of system testing of product requirements (functional, operational, security, safety, and interface) or user acceptance testing of customer requirements (scenarios and qualities of service) over the duration of an iteration.
- **Scenario Details**
  Report provides information on each scenario (completion status, risks, and testing progress)

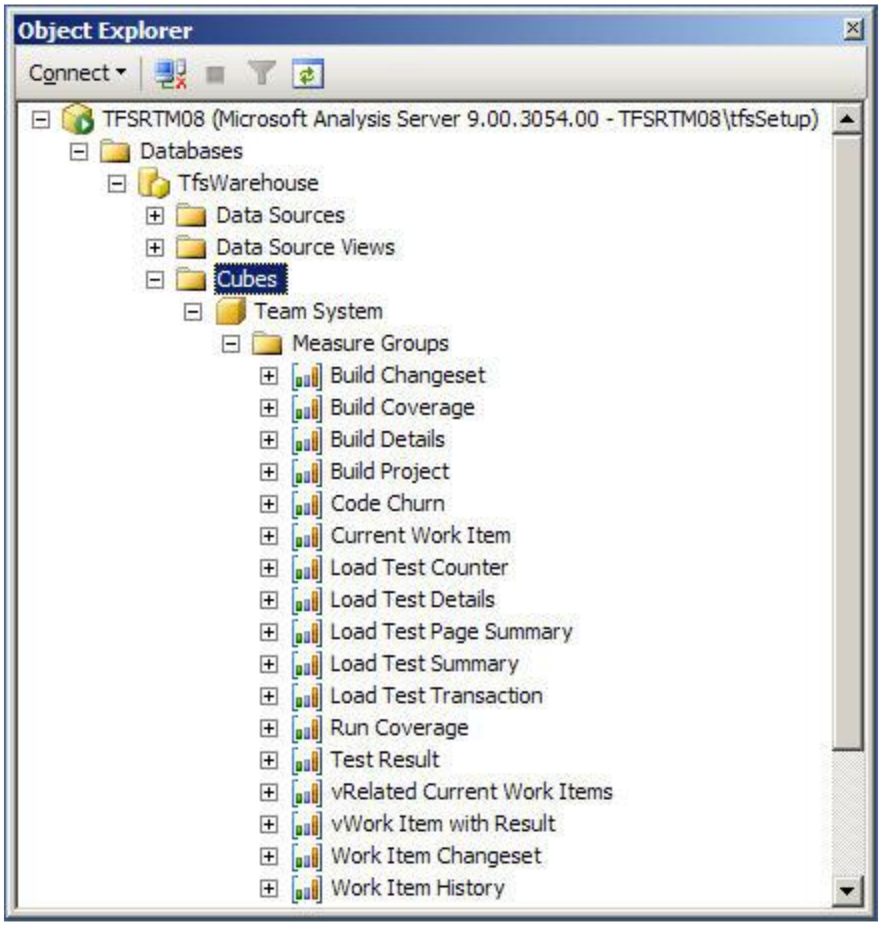
- **Tests Failing Without Active Bugs**
  A listing of tests failing without any active bugs documented
- **Tests Passing With Active Bugs**
  A listing of tests passing with active bugs still present
- **Triage**
  Report shows every work item still in the proposed state and waiting prioritization
- **Unplanned Work**
  A stacked area chart that shows total work versus remaining work, and distinguishes planned from unplanned tasks

Note that each report includes several run-time parameters to scope the output.

BREAKDOWN OF DATA WAREHOUSE TFSWAREHOUSE

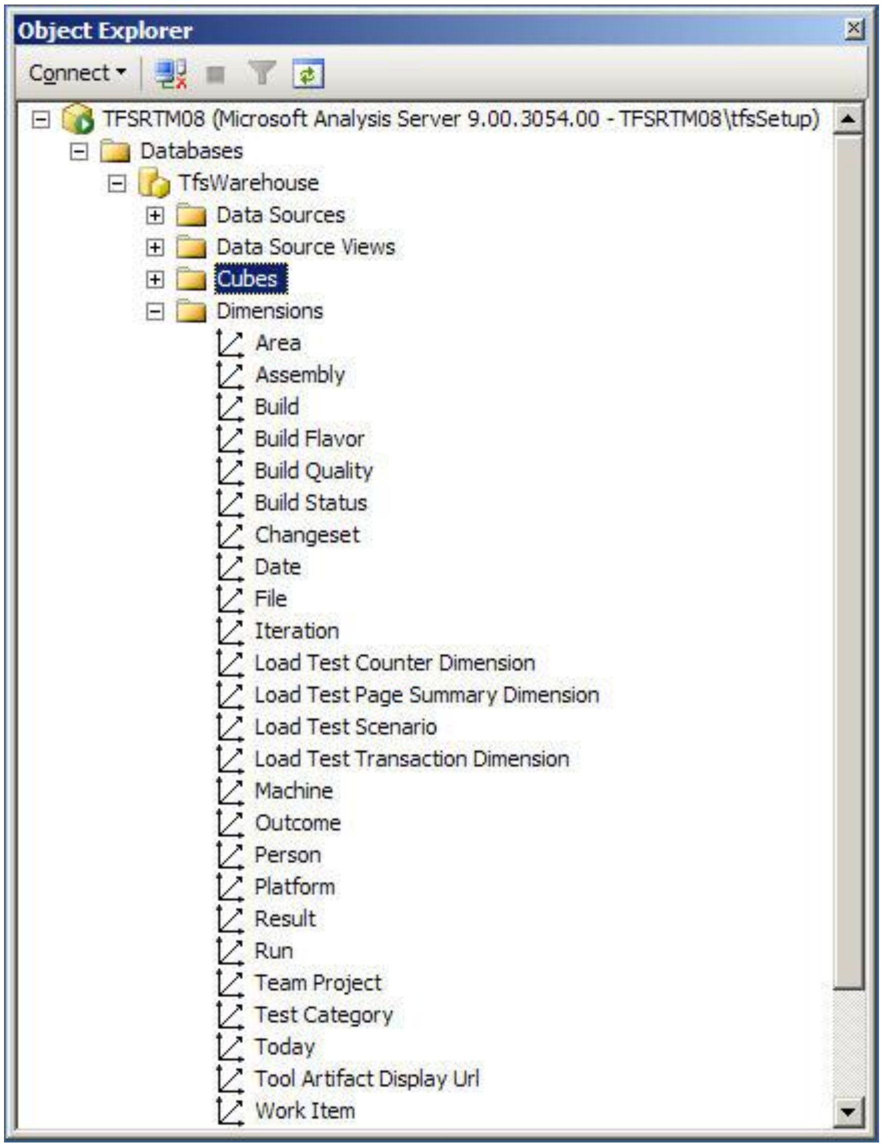
Almost all report data comes from the Team Foundation Server Data Warehouse, called TfsWarehouse. When your development team uses Team System as part of their daily work tasks (e.g. add/change source code and check it in, post bugs, run unit tests, etc.), Team System actually posts this information to the TfsWarehouse. Team Foundation Server collects these activities as data that it can use in reports that reflect the activities of your development team.

Figures 1 and 2 show the complete OLAP cube, TfsWarehouse, in Microsoft Analysis Services. Figure 1 shows the available measure groups, and figure 2 shows the available dimensions that you can use to filter/scope the measure groups.



TFSRTM08 (Microsoft Analysis Server 9.00.3054.00 - TFSRTM08\tfsSetup)
Databases
TfsWarehouse
Data Sources
Data Source Views
Cubes
Team System
Measure Groups
Build Changeset
Build Coverage
Build Details
Build Project
Code Churn
Current Work Item
Load Test Counter
Load Test Details
Load Test Page Summary
Load Test Summary
Load Test Transaction
Run Coverage
Test Result
vRelated Current Work Items
vWork Item with Result
Work Item Changeset
Work Item History

Figure 1: The complete TfsWarehouse OLAP cube.



- TFSRTM08 (Microsoft Analysis Server 9.00.3054.00 - TFSRTM08\tfsSetup)
  - Databases
    - TfsWarehouse
      - Data Sources
      - Data Source Views
      - Cubes
      - Dimensions
        - Area
        - Assembly
        - Build
        - Build Flavor
        - Build Quality
        - Build Status
        - Changeset
        - Date
        - File
        - Iteration
        - Load Test Counter Dimension
        - Load Test Page Summary Dimension
        - Load Test Scenario
        - Load Test Transaction Dimension
        - Machine
        - Outcome
        - Person
        - Platform
        - Result
        - Run
        - Team Project
        - Test Category
        - Today
        - Tool Artifact Display Url
        - Work Item

Figure 2: The complete TfsWarehouse dimensions.

ANALYZING THE QUALITY INDICATORS REPORT AND MAKING DECISIONS

Suppose you've made some staffing changes to your development and QA testing team, just prior to starting your latest project. One of the reasons for the staffing change was because of the increasing number of new bugs with each new build, and a wild variation of failed tests across builds.

As a manager you understand that even the best of development teams could produce a poor build—you are concerned with trends, especially as you get closer to milestones and delivery dates. Over a period of critical builds, you want to know the following:

- Whether more tests are passing than failing
- Whether the number active bugs is going up or down
- How much code is being affected by each build
- The trend of code churn with each changeset
- If the % of code coverage testing has ever fallen below the team's agreed-upon threshold

One of the most popular Team Foundation Server reports is Quality Indicators, a chart that plots these very measures across builds (figure 3). When you run the report, you supply a project as a parameter, and the report will show tests passed/failed, how much code modules have been tested (code coverage), and the extent of code change (code churn). The Quality Indicators report provides an excellent "quick-glance" of the overall status and trend of recent builds.

So, after your team has gone through a few iterations of the following:

- Added a Team System project to Team Foundation Server source control
- Created build definitions
- Defined unit tests
- Released builds to QA, who in turn reported bugs (which were triaged), and then marked addressed bugs as either fixed or not fixed

You can run the quality indicators report for a specific project, Build type, Platform, Flavor, and set of Builds themselves. (You define these parameters when you define the project and create Build Definitions). Figure 3 shows an example of the Quality Indicators report:

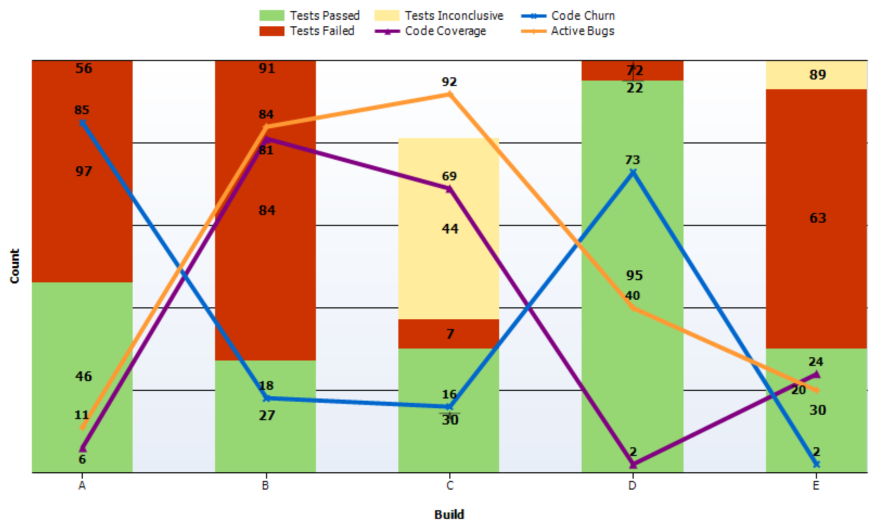


Figure 3: Solution Explorer for an SSRS Report Server project.

Additionally, you can use the information on the Quality Indicators report with your own knowledge of the project to make conclusions. Have you ever heard the old joke that goes, “The operation was a success, but the patient didn’t make it?” You may form conclusions from what you don’t see on the report. For instance, maybe your team hasn’t built enough test definitions into the process. Perhaps your team was distracted by other changes and didn’t have time to define tests in certain areas.

Other times, you may spot delays between large code churn and an increase in bugs–this could be due to Q/A backlog, lack of communication on changes, or other issues.

You can use this Quality Indicators chart, along with your own knowledge of the project and the project team, to visually see if the overall health of your project builds is holding steady, improving, or getting worse.

You may spot a trend from the Quality Indicators report and decide to drill down to details for a particular build. Perhaps the code churn is high and regression testing is finding new bugs: therefore, you might want to look at reports that show regression testing data. Additionally, you want to make sure that your code coverage testing is commensurate with any increase in code churn.

You may see an increase in active bugs as you draw closer to a milestone date, and you may decide to analyze the Remaining Work items report to see if you have the resources necessary to meet the milestone date. Or, you may be surprised by the high number of tests failed for a build and decide to run the Tests Failing report to try to zoom in on specifics for the failing tests.

Team Foundation Server reports contain a number of “pairs” of summarized charts and detailed report listings—helping you to first view trends and then drill down to see supporting details.

WORKING WITH/MODIFYING TEAM FOUNDATION SERVER REPORTS

Modifying Existing Reports

No matter how much you may like “out-of-the-box” reports, you’ll likely want to modify them at some point. Additionally, you may want to create a report that borrows pieces from multiple existing reports. Team Foundation Server reports are “open-architecture” in nature: the reports themselves are SQL Server 2005 Reporting Services files (i.e., they all have an RDL extension). If you need to modify a report (from something as simple as changing a report header to something more involved, such as adding measures to a report), the steps are fairly easy. Essentially, you need to do the following:

1. First, create a Report Server project
2. Second, in the project, create a Data Source for the TfsWarehouse OLAP database (again, all Team System activities are ultimately posted to the Data Warehouse, so you’ll always need to use TfsWarehouse), and retrieve any of the reports you want to change
3. Third, make the necessary report changes. Your changes might be data changes, changes to report input parameters, or report layout changes.
4. Finally, publish the report to the Report Server that Team Foundation Server uses when users run the report

Creating the Report Server Project

Launch SQL Server 2005 Business Intelligence Development Studio, and create a new Report Server Project. For demonstration purposes just call the project Team Foundation Server Reports. You can store the project in the location best suited for your development environment.

Defining the Report Data Source

Second, you need to create a Data Source for any reports you retrieve. The Team Foundation Server reports use both the TfsWarehouse OLAP database from Analysis Services, along with the “regular” SQL Server TfsWarehouse database. Right-click on the Shared Data Sources folder in Solution Explorer (Figure 3), and create the two data sources, one at a time. You must call them TfsReportDS (for the regular database), and TfsOlapReportDS (for the OLAP Database).

Retrieve any of the reports you wish to modify. (Since you’re going to modify at least one of them, you may wish to just retrieve all of them now, even if you only plan to modify one or two for now). Right-click on the Reports folder in Solution Explorer, and select “Add…Existing Item” to navigate to the stored RDL.

As of this writing, the RDL files are stored in a zip file in either of the two locations, depending on whether you are using the MSF for Agile Software Development template or the MSF for CMMI Process Improvement template:

C:\Program Files\Microsoft Visual Studio 2008 Team Foundation Server\TF Setup\MsfAgile_new.zip\Reports

C:\Program Files\Microsoft Visual Studio 2008 Team Foundation Server\TF Setup\1033\MsfFormal_new.zip

After you retrieve the report (in this example, I've retrieved the Quality Indicators Report), Solution Explorer should look like Figure 4.

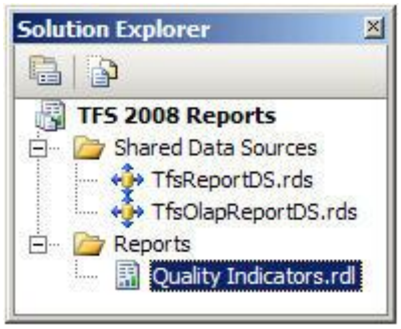


Figure 4: Solution Explorer for an SSRS Report Server project.

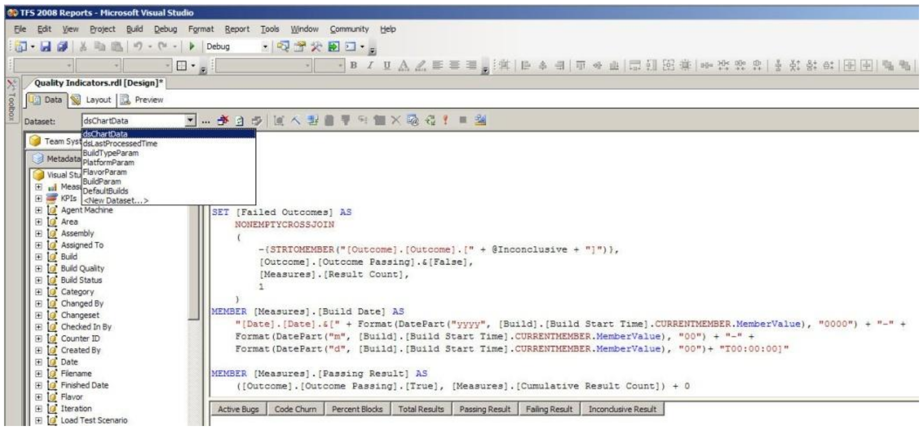
Making Changes to the Report

You can now open the report and make the necessary changes. Here are some definite "things to know" about modifying the reports, which usually falls into three categories: changing data, changing parameters, and/or changing output.

Changing Data on the Report

As the OLAP database TfsWarehouse drives most of the report content, the SSRS report uses MDX to query the database. In a nutshell, MDX is to OLAP databases as T-SQL is to relational databases. At the end of this article, I'll include some recommended references for MDX programming.

Figure 5 shows the Data tab of the SSRS 2005 designer. Note the DataSet pull-down—this lists every set of data that the report uses (either for actual content, or to drive the content of report parameter pull-downs). Each DataSet contains MDX code to retrieve the necessary measures, often scoped to whatever filters/dimensions the report utilizes.



Dataset: dsChartData

- dsChartData
- dsLastProcessedTime
- BuildTypeParam
- PlatformParam
- FlavorParam
- BuildParam
- DefaultBuilds
- <New Dataset...>

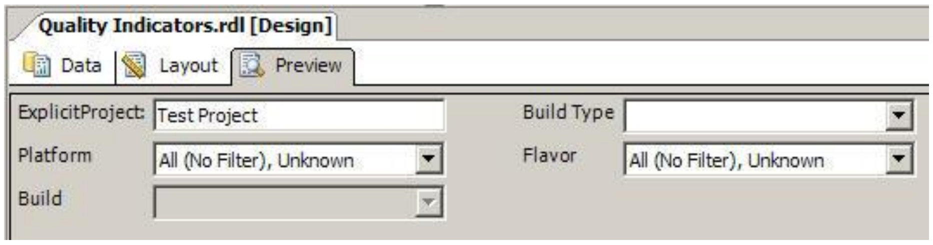
```
SET [Failed Outcomes] AS
    NONEMPTYCROSSJOIN
    (
        -{STRTOMEMBER("[Outcome].[Outcome].[" + @Inconclusive + "]")},
        [Outcome].[Outcome Passing].&[False],
        [Measures].[Result Count],
        1
    )
MEMBER [Measures].[Build Date] AS
    "[Date].[Date].&[" + Format(DatePart("yyyy", [Build].[Build Start Time].CURRENTMEMBER.MemberValue), "0000") + "-" +
    Format(DatePart("m", [Build].[Build Start Time].CURRENTMEMBER.MemberValue), "00") + "-" +
    Format(DatePart("d", [Build].[Build Start Time].CURRENTMEMBER.MemberValue), "00")+ "T00:00:00]"

MEMBER [Measures].[Passing Result] AS
    ([Outcome].[Outcome Passing].[True], [Measures].[Cumulative Result Count]) + 0
```

Figure 5: The Data tab of a Team Foundation Server report, showing DataSets and MDX code.

Changing Parameters on the Report

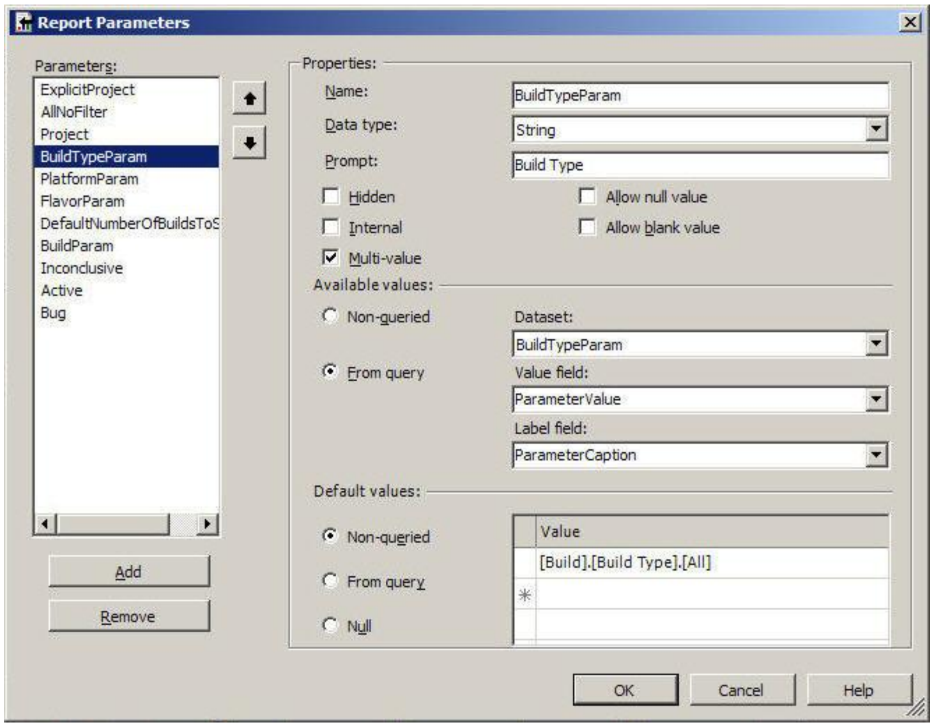
Most of the Team Foundation Server reports prompt for several parameters to scope the output of the report. For instance, the Quality Indicators chart prompts for the following parameters (the project, build type, platform, etc.) (Figure 6).



ExplicitProject:	Test Project	Build Type	
Platform	All (No Filter), Unknown	Flavor	All (No Filter), Unknown
Build			

Figure 6: User-specified parameters for the Quality Indicators Report.

While a few parameters are free-form text, most parameters are pull-down lists that are driven by DataSet parameters. To see the definitions for these parameters, click on the main Reports pull-down option in the SSRS Layout designer and select the option for Report Parameters. You will see all of the parameters, along with options to define any of the parameter data sources (Figure 7).


Report Parameters

Parameters:
- ExplicitProject
- AllNoFilter
- Project
- BuildTypeParam
- PlatformParam
- FlavorParam
- DefaultNumberOfBuildsToS
- BuildParam
- Inconclusive
- Active
- Bug

Properties:

Name: BuildTypeParam
Data type: String
Prompt: Build Type

[ ] Hidden [ ] Allow null value
[ ] Internal [ ] Allow blank value
[✓] Multi-value

Available values:
( ) Non-queried
(●) From query
Dataset: BuildTypeParam
Value field: ParameterValue
Label field: ParameterCaption

Default values:
(●) Non-queried
( ) From query
( ) Null

Value
[Build].[Build Type].[All]

Figure 7: Report Parameters.

You can see the connection to define a data-driven report parameter:

1. First create a DataSet in the Data tab to define the data.
2. Second, create a report Parameter.
3. Third, in the report Parameter, specify the name of the DataSet that provides the data for the parameter. Also note (Figure 6) that you can define a parameter to contain multiple values—this is helpful if a user needs to run a report for multiple selections.

Changing the Layout of a Report

Finally, you can go to the Layout tab to make any physical changes to the layout of the report. The change might be something as simple as adding or changing a label, or something as complex as changing the nature of a chart.

There are countless ways that you might change the layout of a report—far too many to consider all of them here. However, when a report contains a single large control (such as a chart control for the Quality Indicators report), it's important to have a general understanding of the configuration of the chart. Figure 7 shows the chart options for the Quality Indicators chart: a column chart that plots test aggregation counts as stacked bars, and plots other measures as lines. This type of output is helpful to visually analyze how the different measures potentially relate to each other. For each measure, you click on the value name in Figure 8 and select Edit to define the actual plot type and other plotting display options.

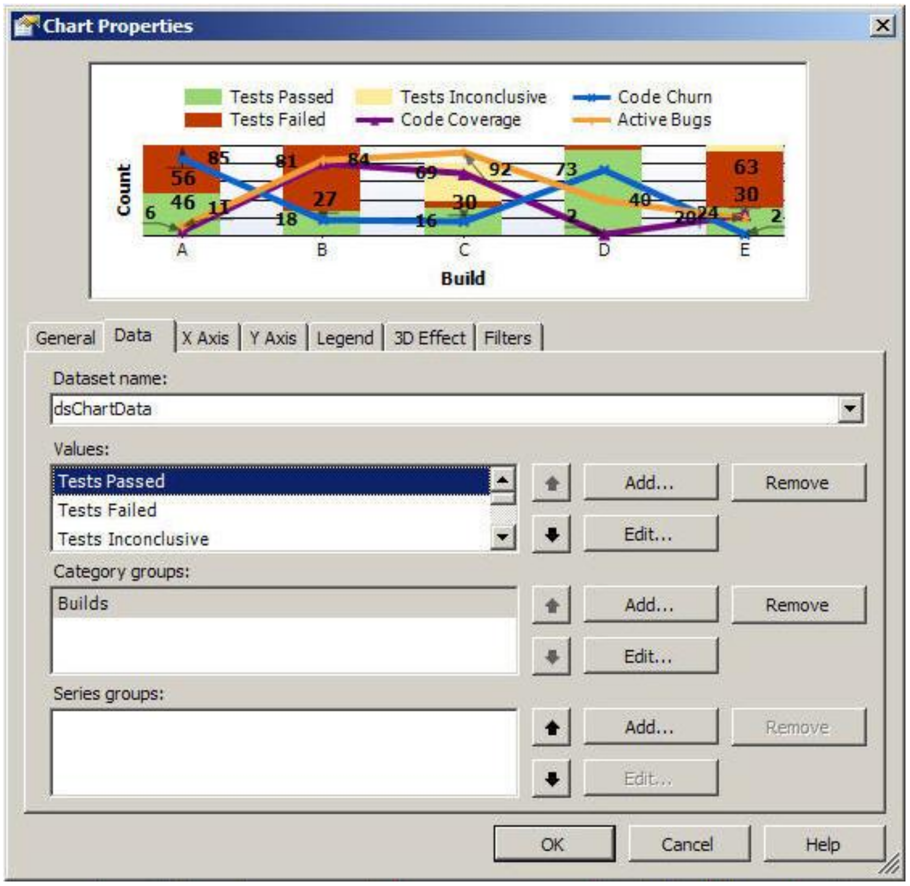


Chart Properties

General | Data | X Axis | Y Axis | Legend | 3D Effect | Filters

Dataset name: dsChartData

Values:
Tests Passed
Tests Failed
Tests Inconclusive

Category groups:
Builds

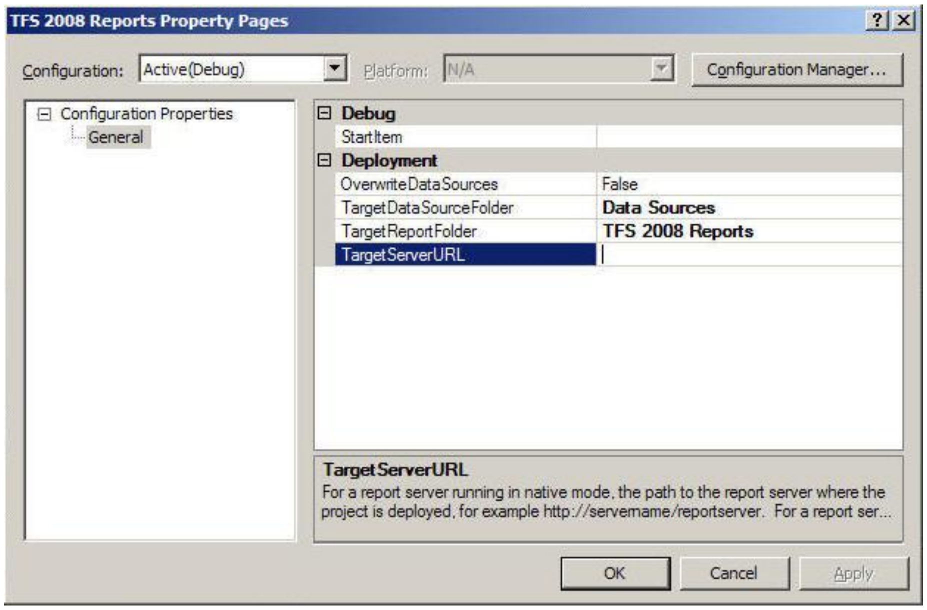
Series groups:

Figure 8: Chart Options for the Quality Indicators Report

So what might be an example of a modification to this report? You might want to display information on the failed tests, or information on the active bugs, etc. In this case, you would go to the layout and add a new report Table control underneath the chart, and drag data from the report DataSet containing the necessary information. (Note that the level of detail you want to display may not exist, either in part or in full, in any of the existing DataSets. In that case, you'll likely need to add some MDX code to retrieve the desired information.

Publishing Changes to the Report

You're almost there! The final step is to publish the changes to the report. You'll need to determine the name and location of your SSRS report server, and then enter those values into the Report Server Project options screen (Figure 9):



Debug	
StartItem	
Deployment	
OverwriteDataSources	False
TargetDataSourceFolder	**Data Sources**
TargetReportFolder	**TFS 2008 Reports**
TargetServerURL	

TargetServerURL
For a report server running in native mode, the path to the report server where the project is deployed, for example http://servername/reportserver. For a report ser...

Figure 9: Report Server Property Page, with TargetReportFolder and TargetServerURL for deployment.

After you specify the TargetReportFolder and TargetServerURL, you can right-click on the project and specify "Deploy". Deployment is usually very fast, and you'll be able to immediately see the effect of your changes!

Writing Custom MDX Against the Data Warehouse

Figure 4 showed an excerpt of MDX code for querying the TfsWarehouse OLAP database. As outlined earlier in this article, MDX is the programming language for querying SQL Server Analysis Services OLAP databases. You'll need to have a good understanding of MDX if you want to build your own custom Team Foundation Server reports.

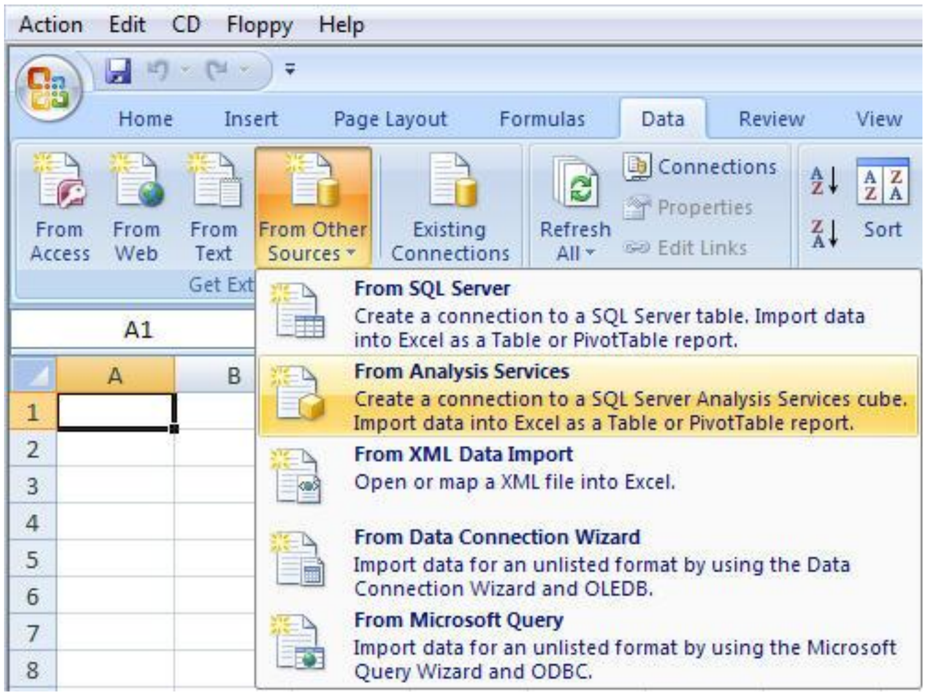
Since the TfsWarehouse OLAP environment is really no different than any other general OLAP environment, you may find yourself wanting to extend the reporting capabilities programmatically. That is where MDX comes in. Here are some areas where you might want to use MDX:

- Add new custom calculations to the TfsWarehouse OLAP cube.
- Create custom MDX named sets for any special report requirements.

Creating New Team Foundation Server Reports Using Microsoft Excel 2007

The classic line in many OLAP environments is that the ultimate Business Intelligence reporting tool is Microsoft Excel. You or your managers may prefer to create new reports/charts against the TfsWarehouse using Excel 2007. The steps to accomplish this are very easy.

First, from the main Excel 2007 toolbar, create a Data Connection to Microsoft Analysis Services (figure 10):



From SQL Server
Create a connection to a SQL Server table. Import data into Excel as a Table or PivotTable report.

From Analysis Services
Create a connection to a SQL Server Analysis Services cube. Import data into Excel as a Table or PivotTable report.

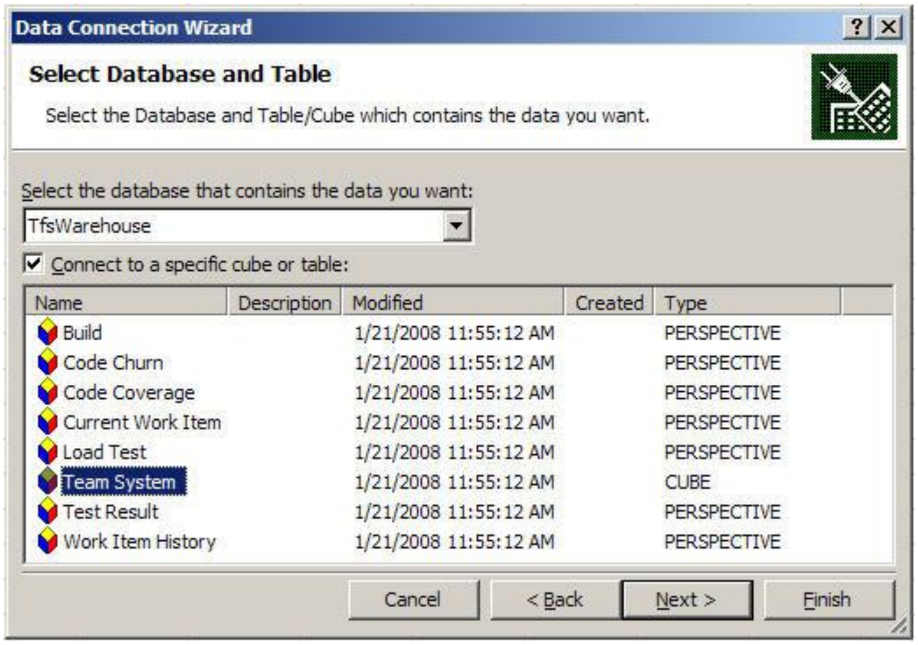
From XML Data Import
Open or map a XML file into Excel.

From Data Connection Wizard
Import data for an unlisted format by using the Data Connection Wizard and OLEDB.

From Microsoft Query
Import data for an unlisted format by using the Microsoft Query Wizard and ODBC.

Figure 10: Creating a Data Connection.

Next, Excel will prompt you for the server name where the TfsWarehouse OLAP database resides. After you provide the server name (and any login/authentication information), Excel will prompt you for the OLAP database and the cube in figure 11 (in case the server contains more than one OLAP database):



Data Connection Wizard

Select Database and Table

Select the Database and Table/Cube which contains the data you want.

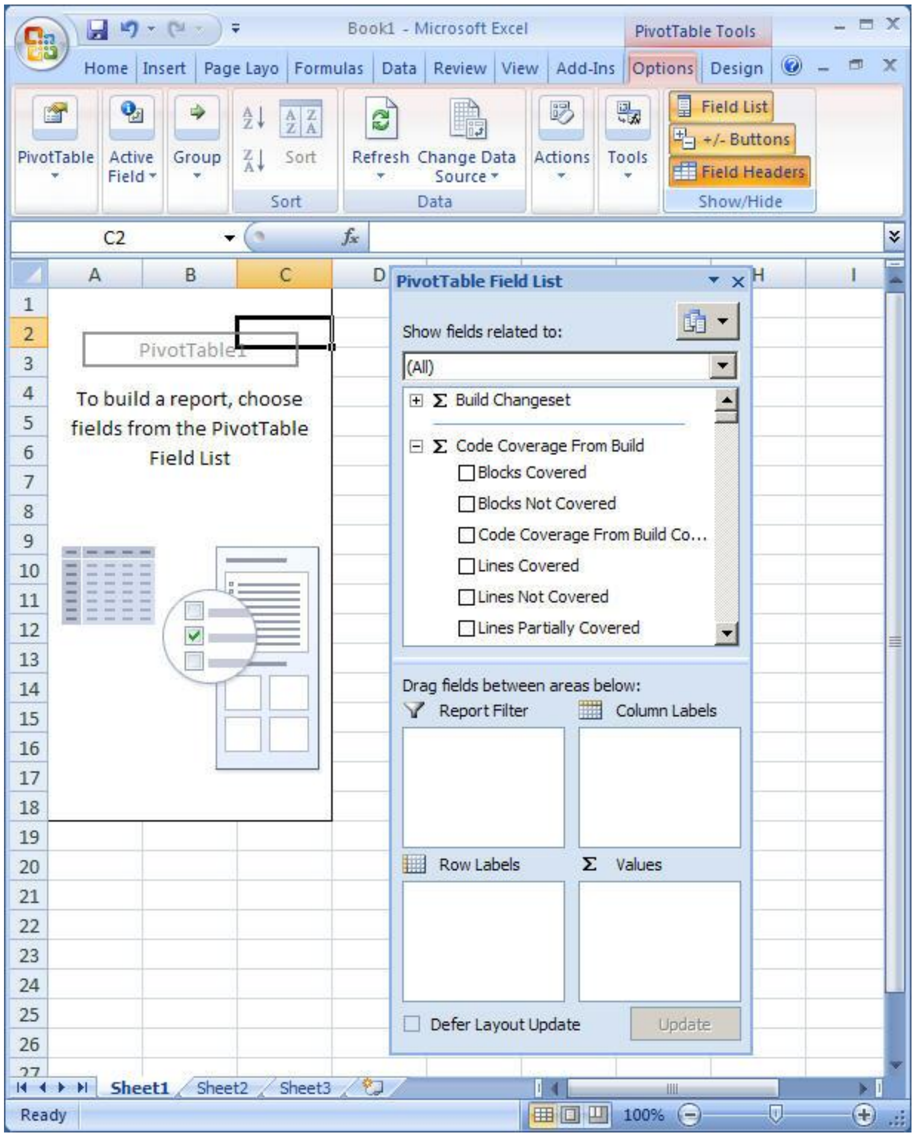
Select the database that contains the data you want: TfsWarehouse

☑ Connect to a specific cube or table:

Name	Description	Modified	Created	Type
Build		1/21/2008 11:55:12 AM		PERSPECTIVE
Code Churn		1/21/2008 11:55:12 AM		PERSPECTIVE
Code Coverage		1/21/2008 11:55:12 AM		PERSPECTIVE
Current Work Item		1/21/2008 11:55:12 AM		PERSPECTIVE
Load Test		1/21/2008 11:55:12 AM		PERSPECTIVE
Team System		1/21/2008 11:55:12 AM		CUBE
Test Result		1/21/2008 11:55:12 AM		PERSPECTIVE
Work Item History		1/21/2008 11:55:12 AM		PERSPECTIVE

Figure 11: Select the TfsWarehouse OLAP database and the Team System cube.

Finally, Excel will create a PivotTable work area and a PivotTable field list. You can then drag measures from the TfsWarehouse OLAP database onto the PivotTable work area (figure 12). You can also create charts as well. Finally, when you are finished, you can save the spreadsheet as a standard Excel file, or you can publish the spreadsheet to Excel Services for eventual deployment to SharePoint.



To build a report, choose fields from the PivotTable Field List

PivotTable Field List

Show fields related to: (All)

- Build Changeset
- Code Coverage From Build
  - Blocks Covered
  - Blocks Not Covered
  - Code Coverage From Build Co...
  - Lines Covered
  - Lines Not Covered
  - Lines Partially Covered

Drag fields between areas below:

Report Filter | Column Labels | Row Labels | Values

Defer Layout Update | Update

Figure 12: Available field list from the OLAP database.

CONCLUSION

As team members use Team Foundation Server, their activities get fed into a SQL Server OLAP Data Warehouse. That activity data is a valuable source of information about that project. Team Foundation Server enables a manager to turn the information from the data warehouse into customizable reports. It's important to understand how Team System tracks development team activities and stores these activities in the data warehouse and how a development/project manager can use reports to analyze team performance. For a manager who needs to make real-time decisions as a project strives to reach milestones, it's worth your time and effort to explore how Team Foundation Server enables the team to capture nearly all of a team's activities. It's important to study the Team Foundation Server Data Warehouse to see what information items are stored in the database, which items are important to your project's success, and then generate reports that drive critical business decisions.

References

- You can find more information on working with Team Foundation Server reports at http://msdn.microsoft.com/library/bb906045.
- For more information on writing MDX queries, I recommend the book Fast Track to MDX, by Mark Whitehorn, Robert Zare, Mosha Pasumansky.
- I've also written an article for CoDe Magazine on MDX, at http://www.code-magazine.com/Article.aspx?quickid=0801051.
- For more information on integrating Reporting Services 2005 with SharePoint, you can read the following white paper, at http://technet2.microsoft.com/windowsserver/WSS/en/library/61b0a571-e337-4c98-b9da-19394682f49c1033.mspx?mfr=true.
- Additionally, the following blog also covers Reporting Services 2005 deployment to SharePoint: http://blogs.sqlxml.org/bryantlikes/articles/628.aspx

ABOUT THE AUTHOR

Kevin S. Goff is the principal consult for Common Ground Solutions (www.commongroundsolutions.net), and has been a Microsoft .NET/C# MVP from 2005 to 2008. He is a .NET and SQL Server consultant, developer, trainer, and writer, and can be reached via his blog at www.TheBakersDozen.net, and via email at kgoff@commongroundsolutions.net.

This white paper was developed in partnership with A23 Consulting.