



Software Testing with Visual Studio 2010



Microsoft
Visual Studio

Jeff Levinson

Praise for *Software Testing with Visual Studio 2010*

“Jeff Levinson has written a remarkable book. Remarkable because it distills a massive amount of information into a clear, readable book that will teach you how to best use the Visual Studio 2010 Testing Tools. Remarkable because it shows not just how to use the strengths of the tools, but also how to work around any of their weaknesses. Remarkable because Jeff walks you through the implementation strategies that can bring real business value, not just to the testing team, but also to test the entire organization. If you are implementing the test tools, this book belongs on your desk. My dog-eared and marked-up copy sits on mine.”

—Steven Borg, Owner, Northwest Cadence

“Testing—and testers—don’t get enough respect. By providing a great mix of the what, why, and how of testing with Visual Studio 2010, this book will help change that. More important, it will help make the software we use better.”

—David Chappell, Principal, Chappell & Associates

“Jeff has once again written a great book, filled with nice nuggets of testing wisdom. A great addition to your testing and ALM library for anyone using Visual Studio 2010 and Team Foundation Server 2010.”

—Mickey Gousset, Microsoft ALM MVP and Senior
Technical Developer, Infront Consulting Group

“Jeff’s book is by far the most in-depth investigation of the Test features in Visual Studio ALM I have seen. His insight and experience help the readers understand the impact of poor testing and how they can improve the quality of their software. I particularly liked the obvious real-world understanding of the realities of software testing when applied in practice and the effort by the author to show the readers the ways around those realities.”

—Martin Hinshelwood, Visual Studio ALM MVP
and Visual Studio ALM Ranger

“Software Testing defines much more than the usage of a testing tool; it shows the practical way in which we test at Microsoft Corporation. Additionally, this book provides the definitive process to using Microsoft Test Manager with the rigor that we test here at Microsoft.”

—Randy Miller, ALM Architect, Microsoft

“Jeff provides the rare combination of deep, insider knowledge of Microsoft’s 2010 testing tools coupled with pragmatic details about how to plan, manage, and execute testing in the real world.”

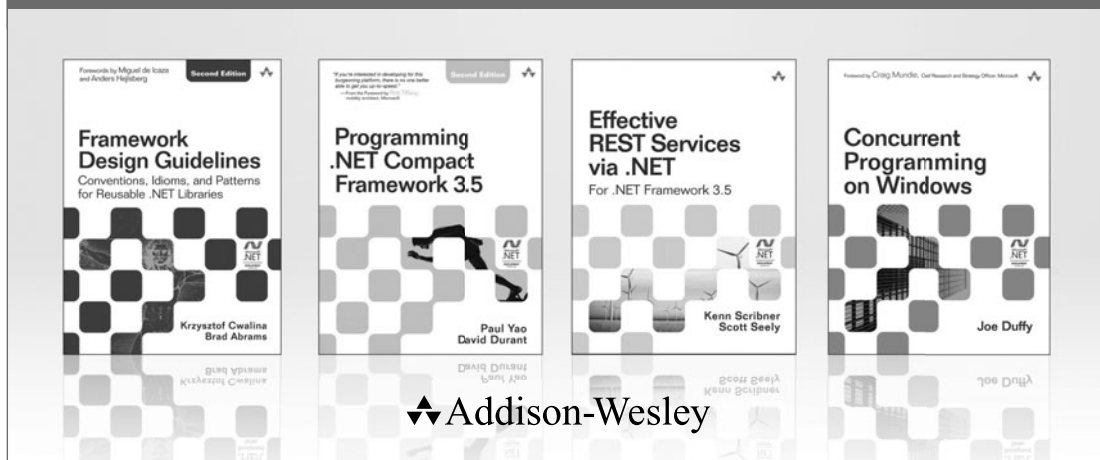
—Mark Mydland, *Director of Test, Visual Studio Ultimate, Microsoft*

“With Jeff’s extensive knowledge with Microsoft’s ALM offering, this book will get you started on the right track with all the new testing capabilities offered by the Visual Studio 2010 suite. Whether you are a new or veteran tester, the personal insights the author brings to the testing topic are very interesting and useful....”

—Etienne Tremblay, *Microsoft ALM MVP*

Software Testing with Visual Studio® 2010

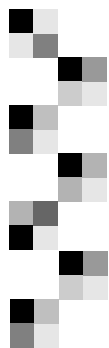
Microsoft® .NET Development Series



Visit **informit.com/msdotnetseries** for a complete list of available products.

The award-winning **Microsoft .NET Development Series** was established in 2002 to provide professional developers with the most comprehensive, practical coverage of the latest .NET technologies. Authors in this series include Microsoft architects, MVPs, and other experts and leaders in the field of Microsoft development technologies. Each book provides developers with the vital information and critical insight they need to write highly effective applications.





Software Testing with Visual Studio® 2010

■ Jeff Levinson

◆◆ Addison-Wesley

Upper Saddle River, NJ • Boston • Indianapolis • San Francisco
New York • Toronto • Montreal • London • Munich • Paris
Madrid • Cape Town • Sydney • Tokyo • Singapore • Mexico City

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and the publisher was aware of a trademark claim, the designations have been printed with initial capital letters or in all capitals.

The .NET logo is either a registered trademark or trademark of Microsoft Corporation in the United States and/or other countries and is used under license from Microsoft.

Microsoft, Windows, Visual Studio, Visual Basic, Visual C#, and Visual C++ are either registered trademarks or trademarks of Microsoft Corporation in the U.S.A. and/or other countries/regions.

The author and publisher have taken care in the preparation of this book, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

The publisher offers excellent discounts on this book when ordered in quantity for bulk purchases or special sales, which may include electronic versions and/or custom covers and content particular to your business, training goals, marketing focus, and branding interests. For more information, please contact:

U.S. Corporate and Government Sales
(800) 382-3419
corpsales@pearsontechgroup.com

For sales outside the United States, please contact:

International Sales
international@pearson.com

Visit us on the Web: informit.com/aw

Library of Congress Cataloging-in-Publication Data:

Levinson, Jeff.
Software testing with Visual studio 2010 / Jeff Levinson.
p. cm.
Includes index.
ISBN 978-0-321-73448-8 (pbk. : alk. paper) 1. Computer software—Testing—Automation. 2. Microsoft Visual studio. I. Title.
QA76.76.T48L48 2010
005.1'4—dc22

2010038104

Copyright © 2011 Pearson Education, Inc.

All rights reserved. Printed in the United States of America. This publication is protected by copyright, and permission must be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise. For information regarding permissions, write to:

Pearson Education, Inc.
Rights and Contracts Department
501 Boylston Street, Suite 900
Boston, MA 02116
Fax: (617) 671-3447

ISBN-13: 978-0-321-73448-8

ISBN-10: 0-321-73448-3

Text printed in the United States on recycled paper at RR Donnelley Crawfordsville in Crawfordsville, Indiana.

First printing February 2011

*To my wife, Tami, and my daughter, Caitlin, who supported
me and had to deal with me for the last year.
And my new son, Sean: I hope you start sleeping through the night soon.*



This page intentionally left blank



Contents at a Glance

Foreword xvii

Preface xxi

Acknowledgments xxix

About the Author xxxi

- 1 State of Testing 1**
- 2 Software Quality and Testing Overview 13**
- 3 Planning Your Testing 29**
- 4 Executing Manual Tests 71**
- 5 Resolving Bugs 107**
- 6 Automating Test Cases 135**
- 7 Executing Automated Test Cases 183**
- 8 Lab Management 209**
- 9 Reporting and Metrics 239**

This page intentionally left blank



Contents

Foreword xvii

Preface xxi

Acknowledgments xxix

About the Author xxxi

1 State of Testing 1

Software Testing Challenges 1

The Need for Testers 3

A Different Approach 5

Fixing Communication 5

Increasing Project Visibility 6

What Are the Tools Designed to Do? 7

Metrics 10

Citations 12

2 Software Quality and Testing Overview 13

Software Quality 13

Requirements 14

Business Value 14

Expectations 15

Nonfunctional Requirements 15

Where Do You Build Quality? 17

Process and Quality 19



Software Testing	19
<i>The Testing Mindset</i>	20
<i>Software Testing Strategies</i>	21
<i>Types of Software Testing</i>	22
<i>Test Management</i>	27
<i>After the Product Is Shipped or Deployed</i>	27
3 Planning Your Testing	29
Microsoft Test Manager	30
<i>Test Plans</i>	36
<i>Properties</i>	38
<i>Contents</i>	43
<i>Adding Suites and Test Cases to Your Plan</i>	46
Testing Configurations	48
<i>Managing Test Configurations</i>	49
<i>Assigning Test Configurations</i>	51
Assigning Testers	53
Test Case Planning Workflow	55
<i>Analysis and Initial Design</i>	56
<i>Construction</i>	61
<i>User Acceptance Testing</i>	62
Common Scenarios	64
<i>Scheduling and Tracking Test Case Creation and Execution</i>	64
<i>Feature Driven Development</i>	65
<i>Moving from One Iteration to Another</i>	67
<i>Handling Different Test Configurations</i>	68
4 Executing Manual Tests	71
Using the Test Case Work Item Type	72
<i>Shared Steps</i>	75
<i>Data Driven Test Cases (Test Parameters)</i>	77
Running Your First Tests	79
<i>Test Runner</i>	80

Examining Test Results	92
<i>Test Run Results</i>	93
<i>Detailed Test Results</i>	95
Exploratory Testing with MTM	101
5 Resolving Bugs	107
A Bug's Life	107
<i>Customer Reported Bug</i>	110
<i>Test Team Reported Bug</i>	110
<i>Triaging the Bug</i>	110
<i>Reactivations</i>	111
Bug Differences and Modifications	111
The Generated Bug	116
How a Developer Uses IntelliTrace	120
Fixing the Bug	122
<i>Associated Changesets</i>	124
<i>Associated Work Items</i>	124
<i>Impacted Tests</i>	125
Setting the Build Quality	125
Assigning a New Build	127
Verifying That the Bug Is Fixed	129
Dealing with Impacted Tests	131
6 Automating Test Cases	135
To Automate or Not to Automate	136
The Automated Testing Framework	139
Creating an Automated Test from a Manual Test	141
<i>Examining a Generated Web Application Coded UI Test</i>	142
Adding Validations	157
Adding Additional Recorded Steps	164
Parameterized Coded UI Tests	166
<i>Handling Issues Due to Inconsistency</i>	168
<i>Resolving the Data Inconsistency</i>	169



Handling Dynamic Values	172
<i>Other Tips</i>	177
Combining Multiple Tests	178
Associating Coded UI Tests and Test Cases	178
7 Executing Automated Test Cases	183
Executing Automated Tests Through Visual Studio	183
<i>Local Execution</i>	184
<i>Local Execution with Remote Collection</i>	184
<i>Remote Execution</i>	185
Executing Automated Tests from the Command Line	190
Executing Automated Tests in MTM	191
<i>Creating an Automated Build</i>	191
<i>Setting Up the Physical Environment</i>	193
<i>Running a Coded UI Test Through MTM</i>	196
Executing Automated Tests with Team Build	202
Automated Testing Gotchas	205
<i>Custom Dialogs</i>	205
<i>Cleaning Up Your Tests</i>	207
8 Lab Management	209
Managing Virtual Environments Through MTM	210
Finishing Virtual Environment Configuration	217
Automated Test Settings	221
<i>Lab Management Workflow</i>	222
Executing a Lab Build	231
Running Automated Tests Through MTM	233
Manual Tests in a Virtual Environment	234
9 Reporting and Metrics	239
Understanding the Reporting Structure	240
Built-In Reports	242
<i>Bug Status</i>	244
<i>Bug Trends</i>	245
<i>Reactivations</i>	246



<i>Build Quality Indicators</i>	246
<i>Build Success over Time</i>	248
<i>Build Summary</i>	249
<i>Stories Overview</i>	250
<i>Test Case Readiness</i>	251
<i>Test Plan Progress</i>	252
Excel Services Reports (Dashboards)	253
Reporting with Microsoft Excel	254
<i>Creating a Generated Report</i>	255
<i>The Testing Measures</i>	256
Metrics	268
<i>What to Measure</i>	271
<i>First-Time Defect Rate</i>	273
<i>Bug Reactivations</i>	276
<i>General Bug Counts</i>	277
 Index	 283

This page intentionally left blank



Foreword

OUR PRODUCT TEAM LIKES TO SAY that when we started building Visual Studio Test Professional 2010 we wanted to deliver world peace as a feature. To make our ship date, we reduced our aspirations to making peace between software developers and software testers.

Even with this drastic reduction in scope, we faced a daunting task. Our profession often creates substantial separation, organizationally and sometimes physically, between those responsible for creating and maintaining software and those responsible for validating that the software meets the needs of businesses and customers. Because of the separation developers and testers often communicate by throwing information “over the wall,” which results in poor communication of issues (bugs); in uncertainty about what features, bug fixes, and improvements development has added to a particular build; and in mistrust between the development and test organizations. All of which, in turn, contribute to the quality issues, schedule delays, and outright project cancellations that continue to plague our industry. Many of the QA tools currently available in the market exacerbate the communication problems by managing the planning, testing, and tracking of the test effort independently from the tools used to track planning and development.

As we began to dig into the source of the communication breakdowns, we found, somewhat to our surprise, that manual black-box style testing accounts for approximately 70 percent of all testing in our industry. To succeed with this style of testing, testers develop deep domain knowledge around the products they test but spend less time cultivating their knowledge



of the deep technical and architectural aspects of the system. To manage their testing efforts, these testers relied largely on Microsoft Word, Microsoft Excel, handwritten notes, and whiteboards. Worse, testers had no tool support for running tests and therefore spent significant portions of the day on time-consuming and often menial tasks such as deploying software, writing bug reports, and finding clear steps to reproduce bugs.

As a software development company, Microsoft clearly recognizes the importance of allowing all members of software development teams, developers, testers, architects, project and program managers, and business analysts to participate fully in the development process. We built Microsoft Visual Studio Test Professional 2010 and the Lab Management capability of Microsoft Team Foundation Server 2010 to help eliminate the friction between developers and testers by providing self-documenting builds that include tests impacted by developers' changes, single-click actionable bugs that eliminate the "No Repro" problem, and work item-based test planning and management that enables visibility and traceability by all project stakeholders. To streamline the test effort and increase the effectiveness of testing, we added streamlined virtual build, deploy and test, fast forward for manual testing, and the capability to generate an automated Coded UI test based on a previously completed manual test.

During development, we relied heavily on feedback and advice from a number of external sources who could provide both industry perspective and feedback based on extensive personal experience. In that capacity, Jeff Levinson helped to shape Microsoft's test offering and TFS's Lab Management capability in just about every way possible. Jeff participated in every formal design review, special interest group, technology preview, and beta program that existed. Beyond the formal interactions, Jeff spent days and weeks of his "free time" installing, using, and testing our product followed by hours spent with me and other members of the team providing feedback, pushing for improvements, and making suggestions. I can't say that all Jeff's feedback made it into the final product, but our product is better for his effort.

As much as I would like to believe that Jeff spent all this time and effort just to make my life easier, I know that Jeff's real motivation came from his passion for helping teams to build quality software. In his book, Jeff brings

a pragmatic approach, years of experience, and a clear understanding of how the entire development team must work together to build truly great software. The combination of Jeff's insider knowledge and deep understanding of Microsoft Visual Studio Test Professional 2010 with his proven approach to software testing create a roadmap that can help any team to build high-quality software while avoiding the pitfalls and friction that so often lead to "death marches," missed deadlines, and unsatisfied customers.

Mark Mydland

Director of Test, Microsoft

This page intentionally left blank



Preface

AS A PROCESS IMPROVEMENT professional, I have experienced many team challenges. Big challenges. It is not unusual to see teams that seem so perfectly compatible start in excitement only to fizzle in different directions and end up not working together. Products suffer, customers suffer, and ultimately relationships suffer. When Microsoft introduced a new set of tools to help ensure quality applications and bring teams together in an evolutionary way, I was at first skeptical, but not now.

From one company to the next, one organization to the next, or even within a given team, the same problems arise. Granted, the circumstances can make basic problems much more challenging, but you can consistently identify the following issues:

- Challenged or poor communication between developers and testers
- Constant churning with precious little progress due to fixing the same things over and over again
- Organizational structures that sabotage quality work and the capability to productively manage resources
- Management that focuses on the shipping date with no consideration of the long-term cost of poor quality
- Lack of proven toolsets to maximize productivity and efficiency of teams

Enter Microsoft Test Manager. Now there is this single point at which teams can coalesce: quality. Microsoft Test Manager offers the following proposition: Do you want to build a better quality product with less rework,



less divisiveness in a shorter period of time? If so, what are you willing to do to achieve this goal? The response seems simple enough:

- Incorporate a basic process with some good old-fashioned common sense.
- Use common tools.
- Share data.

In my experience, there is one obvious set of tools: Team Foundation Server, Visual Studio, and Microsoft Test Manager. Using these tools has been proven to break down barriers, get teams talking, and deliver the promises of the preceding proposition. It is my goal to demonstrate how to accomplish this to as many people as possible. With the tools that Microsoft provides, the level of effort required to use them is minimal and the benefits are huge. Will the tools work for everyone? Well, with the wide variety of tools and platforms that individuals need to test against, I can't make any promises. But if, for the most part, the platforms and languages you test against are somewhat commonplace, you can reap benefits from using this tool suite.

It's funny how we see the process differently depending on our role on a project. I have served in many roles (some better than others). As a developer, I couldn't stand testers because they always broke my code because they didn't know how to use the application. As a tester, I couldn't stand developers because they didn't know how to code. As an architect, I looked on much of the process as a necessary evil. As a process improvement expert, I realized (even though this may be patently obvious) that without testers I couldn't get the metrics I needed to make a difference. As an author, I hope to communicate that by bringing testers and developers together to work *cooperatively* we can make positive changes across the board in a fun and cooperative environment. We can accomplish this by objectively assessing and learning about these unique and valuable new tools from Microsoft.

Thank you for reading this book, and I hope it helps you improve the quality of your software. If you have questions, errata, suggestions, additions, or disagreements with anything you read, please drop me a note at jeffstuff@jtlevinson.com.

Who Should Read This Book?

This book is primarily for software testers or people who test software as one of their primary job roles—from the professional tester or developer to the business analyst who needs to verify software for end users.

The testing process with Microsoft Test Professional 2010 and Visual Studio 2010 Ultimate is structured in a way that the tester can perform manual testing, and the *developer* can automate the tests. For this reason, developers can also find this book useful because considerable resources are dedicated to the developer's role in the testing process. Further, much of this book covers best practices for developers and testers working together. Chapter 6, "Automating Test Cases," and Chapter 7, "Executing Automated Test Cases," are especially relevant to the topic.

For those new, or relatively new, testers Chapter 2, "Software Quality and Testing Overview" provides a solid introduction to the goals of testing, approaches to testing, and considerations when testing. This is designed to be a primer and can be skipped by those already familiar with testing processes.

Test and development managers, in particular those looking for a better understanding of the overall process or those wanting to leverage the reporting offered in Team Foundation Server, can also benefit from reading this book. Understanding reporting is often a conduit for discovering that a seemingly insurmountable problem can actually be fixed. Add to that mix the capability to quantify metrics and improve them over time, and you have a powerful tool for managers. Chapter 1, "State of Testing," Chapter 3, "Planning Your Testing," and Chapter 9, "Reporting and Metrics," are most applicable to managers.

I hope you find this book helpful in your organization and as a guide for your testing teams.

What This Book Does Not Cover

One topic not covered is the virtualization infrastructure required to run Lab Management. The lab infrastructure requires a network administrator and people familiar with virtualization technologies including hardware and software. It would have been too complicated to include everything and would have been beyond the scope of this book. This includes information such as



System Center Virtual Machine Manager and Hyper-V. Setup of the Test Agent, Test Controller, and Build Hosts are discussed because these are items the testing or development team will probably need to deal with at some point—especially if teams switch back and forth between user interface testing and unit testing.

Additionally, you will not find information on load testing, stress testing, and Web performance testing, and only minimal information on unit testing is available. The information on unit testing is presented from the perspective of how a tester or developer might execute these automated tests and relate them to requirements. You will not find any in-depth discussions on the philosophy of unit testing or specific unit testing techniques.

About the Application Used in This Book

The application used throughout this book (and in the exercises) is the BlogEngine.NET application because it is a popular real-world application used by many individuals. It is also open source with a thriving community behind it. BlogEngine.NET was created by Al Nyveldt and Mads Kristensen. You can get more information about it from dotnetblogengine.net and download the original version of this application at blogengine.codeplex.com.

The version used in this book has been modified somewhat because it was ported to Visual Studio 2010 and converted to a Web Application for use with Team Build. You can download the source from here: informit.com/title/9780321734488. This download includes a readme file describing how to set up the application so that you can follow along with the examples. Aside from these modifications, no other material modifications have been made to the source code. The Database project and the MSDeploy project were added to support the different capabilities of the tools demonstrated.

Other software is required to follow these examples. You must have either Visual Studio Ultimate or Premium to create Coded UI tests. You must also have Microsoft Test Professional or Visual Studio Ultimate to get Microsoft Test Manager. Although not a tested configuration, you might run these examples with Microsoft's all-up Lab Management virtual machine. This virtual machine can be downloaded here: www.microsoft.com/downloads/details.aspx?FamilyID=592e874d-8fcd-4665-8e55-7da0d44b0dee&displaylang=en.



How This Book Is Organized

This book is structured to not only be used as a reference but also as a step-by-step guide for a process. The book guides you through the testing process as it would occur on an application. The book starts with a discussion of problems the industry faces and quickly moves to development methodologies and the role of testers. From there, you learn how to plan the testing process using Microsoft Test Manager to write first-draft Test Cases and execute those Test Cases. During the course of execution, bugs can be filed, and developers can fix those bugs. Testers can verify the fix and then determine which Test Cases to automate. Developers automate the Test Cases and then they can be executed by developers and testers in a physical or a virtual environment. Finally, the book ends with a discussion of reporting and metrics and offers some ideas that you can apply to your processes to improve quality.

- Chapter 1, “State of Testing”—This chapter provides an introduction to the problems facing software development teams today from a quality perspective. It covers the cost of poor quality, legal actions because of poor quality, and other commonly known but frequently ignored issues. It also discusses the author’s philosophy of software testing and the goals of this book. Finally, it covers some of the basic software development methodologies and where software testing fits in with these methodologies. This chapter provides an overview of the Microsoft technology stack and end-to-end process flow.
- Chapter 2, “Software Quality and Testing Overview”—This chapter presents an introduction to software testing. This includes why we need to do software testing, what the goals of software testing are, different types of software testing and software testing techniques. It provides a foundational view of the tester’s world.
- Chapter 3, “Planning Your Testing”—First, you must plan for testing. This chapter shows you how to use the tools in Microsoft Test Manager (MTM). It also details how to navigate MTM, create test settings, and structure Test Cases. It explains how to manage the testing process using the Test Case work item type and requirements.

- Chapter 4, “Executing Manual Tests”—This is your introduction to executing tests with Microsoft Test Manager and Test Runner. You learn how to create Test Cases, reuse test steps, execute manual tests, and file bugs. When teams first start using MTM, this is what they do on a day-to-day basis. This chapter also covers exploratory testing.
- Chapter 5, “Resolving Bugs”—When you file a bug, the process and lifecycle of the bug is critical to reducing rework and driving reporting. This chapter discusses the Bug work item type, some customizations you might want to make to it, and how it serves as a communication mechanism for testers and developers. You are also introduced to how developers can use the Bug work item type to fix software correctly the first time and then how testers can verify that fix. This chapter introduces Test Impact Analysis.
- Chapter 6, “Automating Test Cases”—This is largely a developer-focused chapter on creating automated Test Cases. These Test Cases can be manual (recorded by testers) or automated (unit testing and other types of tests). One key item in this chapter is associating any type of automated test with a requirement. The features in this chapter require Visual Studio 2010 Premium or Ultimate.
- Chapter 7, “Executing Automated Test Cases”—After automating the Test Cases, teams need to execute those automated tests. This chapter describes all possible ways to execute an automated test. This is both a developer-focused chapter (using Visual Studio to execute the tests) and a tester-focused chapter (using Microsoft Test Manager to execute the tests). You also learn how to execute tests as part of the build process.
- Chapter 8, “Lab Management”—This chapter focuses on the Lab Management features of Microsoft Test Professional 2010 and Visual Studio 2010 Ultimate. You learn how to use the virtualization platform to test applications and how to snapshot environments to help developers reproduce bugs. This chapter also focuses on both developers and testers: Developers learn how to set up the code to be deployed in a lab environment and executed through the build process. Testers learn



how to execute both manual and automated tests in a virtual environment and file actionable bugs.

- Chapter 9, “Reporting and Metrics”—This chapter covers reporting and metrics. By following the processes in this book, a team has detailed metrics by which they can determine the quality of their product and where the testing process can be improved. You explore the built-in reports provided by TFS but also how to do custom reporting on the data cube to get detailed information about your testing endeavors. This also serves as a brief guide to what type of customizations you can make to the Test Case work item type to capture more detailed information.

This page intentionally left blank



Acknowledgments

WRITING THIS BOOK HAS been a labor of love over the last year, and there is no way I could have done it alone. Writing about beta software requires cooperation from the teams at Microsoft, and in this case I got more than I could have imagined. The number of people who provided input is long. If I have left anyone out, my apologies. First a special thanks to Mark Mydland, Ram Cherala, and Euan Garden—all of them put up with me for what seemed like hours on end. Long before starting the book, I pestered them on the philosophy of testing and their approaches to it and tools to implement it. I learned a lot from all of them. Other members of the testing team helped with everything from technical aspects to the understanding of specific decisions made during the creation of Microsoft Test Manager, Lab Management, and the Coded UI features. Many thanks to Naysawn Nadiri, David “Dr. Will” Williamson, Dominic Hopton, Pradeep Narayan, Ravi Shanker, Chris Patterson, Anutthara Bharadwaj, Daryush Laqab, Shay Mandel, Vinod Malhotra, Gautam Goenka, Vijay Machiraju, and Mathew Aniyam.

One other group of individuals helped as well, whether they knew it—the Application Lifecycle Management (ALM) Most Valuable Professionals (MVPs). They put up with hundreds of e-mails and provided responses that helped shape my approach to using the testing tools. Many were supportive during the writing process. I am honored to be included in this exceptionally talented and knowledgeable group of people.



On a personal note, my wife Tami and daughter Caitlin have had to put up with an absentee husband and father for the last several months. Needless to say I could not have done this without their love and support.

My four reviewers deserve a big thank-you. Mario Cardinal, Etienne Tremblay, and Mike Vincent are fellow ALM MVPs who thoroughly vetted my content not once but twice. The book is better for their input. And to Anutthara Bharadwaj (a member of the test team), even after a long plane flight with no power and bad service, she provided excellent feedback and comments and continued to teach me even after I thought I “knew it all.” Thank you, Anu, for putting up with me.

I want to call out Mike Vincent specifically here for not only his help but his contributions. Mike was the last reviewer of this book. During the production of this book many ideas were discussed as to what this book should be about and what it should cover. Late in the process we decided that Chapter 2 should be added to provide an introduction for those just coming into the testing space. I did not have time at that point in the schedule to write this chapter. In came Mike to the rescue. Chapter 2 was contributed by Mike and helps round out the book in a way that makes it better than it was before. Thanks Mike!

For Brian Keller, a senior technical evangelist with Visual Studio, I can only say “Thank you.” I am just in awe of his ability to quickly read, distill, and correct information or add the tiny details that were missed. And to Sam Guckenheimer for helping iron out the most difficult part of any technical book—the direction.

And to my co-workers at Northwest Cadence who were supportive of the entire process from beginning to end and through the days, weeks, and months of the writing process.

Most important, thanks to my editor Joan Murray and assistant editor Olivia Basegio, without whom this book would not have been possible. As with any endeavor, it isn’t the big stuff that trips you up; it’s the small stuff. And to the rest of the team at Addison-Wesley from San Dee Phillips my copy editor to Andrew Beaster who shepherded the book through production twice, thank you for sticking with it!



About the Author

Jeff Levinson has 16 years of experience in software development in many different roles—developer, tester, architect, project manager, and scrum master at several large companies. Jeff is currently a Senior Application Lifecycle Management (ALM) Consultant for Northwest Cadence, which is a company that specializes in Team Foundation Server, Visual Studio, methodologies, and process improvement. In his day-to-day work, Jeff helps teams, organizations, and companies adopt more efficient processes, improve quality, and reduce costs associated with software development. Jeff is a frequent speaker at industry events and writes a twice-monthly column for *Visual Studio Magazine Online*. This is his fourth book on software development. His other books are *Building Client/Server Applications with VB.NET* (Apress 2003), *Pro Visual Studio Team System 2005* (Apress 2006), and *Pro Visual Studio Team System with Team Edition for Database Professionals* (Apress 2007). Jeff has a master's degree in software engineering from Carnegie Mellon University and is an MCP, MCAD, MCSD, MCDBA, and MCT.

Jeff currently lives in Washington State with his wife and two children.

This page intentionally left blank

3

Planning Your Testing

TO BEGIN, YOU NEED A PLAN. The plan does not need to be 500 pages of documentation or a massive Gantt chart. This chapter covers how to create a Test Plan with Microsoft Test Manager (MTM) and the various options that the Test Plan provides to you. More important, this chapter covers *what* to test and how to get involved as a tester early in the development process. In addition, Microsoft provides a little-used Test Plan Word template that can help answer some questions about the testing process up front.

Another key item covered here is how to plan and test for multiple iterations. Can you reuse your Test Cases, and does it make sense to do that? Many items come into play when planning the testing for an entire release versus a single iteration. By the end of this chapter, you will know how to use the Plan tab of MTM, create new plans, and create a framework for testers to work in for a given period of time.

As mentioned in Chapter 1, “State of Testing,” testers should be involved, ideally, during the requirements gathering process. In a waterfall cycle this is during the Analysis phase. In an agile cycle this is during the period of time in which the business analyst or product owner fills in the details for items on the Product Backlog but before introducing the item into an Iteration Backlog. This chapter covers what the testers’ responsibilities should be and what they can do to help reduce the potential for bugs to be introduced into the software.

TEST APPROACH

When starting any testing endeavor, you need an approach to the process. Consider what is acceptable, what are the criteria for release, how you can perform each type of test, and other information that forms the framework of the approach. If you use the MSF for Agile v5.0 process template, there is a Test Approach Word template located in the sample documents on the SharePoint site. (The path to the document is Documents/Samples and Templates/Test/Document Template - Test Approach.dotx.) You can also find a sample document showing how the Test Approach looks when filled out.

Microsoft Test Manager

Microsoft provides a separate tool for testers: Microsoft Test Manager (MTM) where you can create Test Plans and add and update Test Cases and where manual and automated tests are executed from. Before getting into the details of creating Test Plans, you need to understand how to navigate within MTM. Figure 3-1 shows the navigation controls.

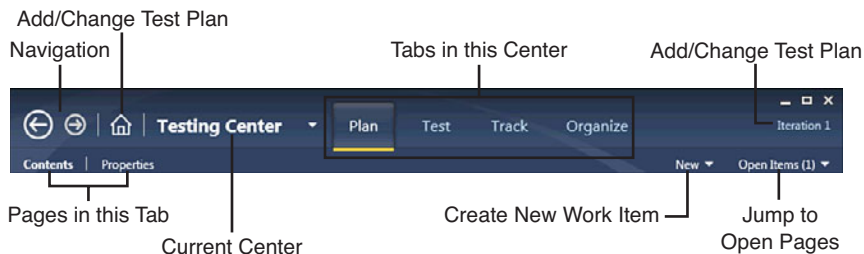


FIGURE 3-1: MTM navigation controls

MTM is organized into Centers, Tabs, and Pages, as shown in Figure 3-2.

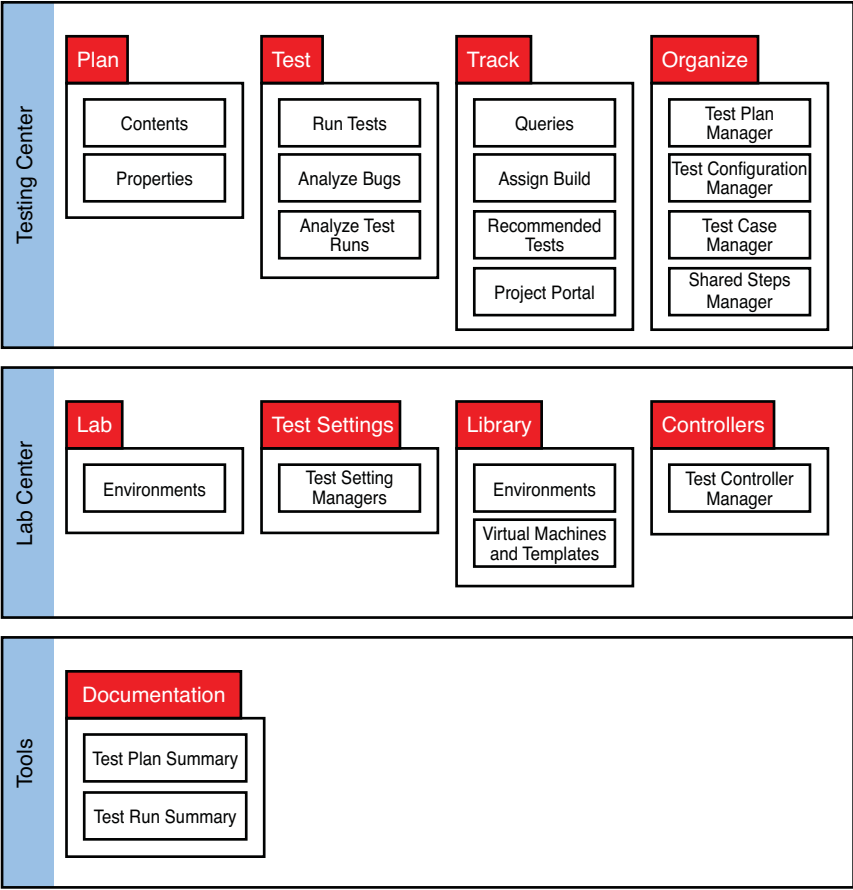


FIGURE 3-2: Microsoft Test Manager navigation layout

Table 3-1 briefly describes each section. These pages and the options they enable are described throughout the book.

TABLE 3-1: MTM Pages Described

Center	Tab	Page	Description
Testing	Plan	Contents	Contains the settings for the given Test Plan including manual and automated test settings, test configurations, and the build in use
		Properties	Contains the suites and Test Cases that need testing for the selected plan



TABLE 3-1: Continued

Center	Tab	Page	Description
	Test	Run Tests	Main page for executing test runs
		Verify Bugs	Contains bugs that have been resolved that the tester can quickly get to and verify
		Analyze Test Runs	Shows all test runs (manual and automated) but used mainly to view an automated test run and take appropriate actions based on the outcome of the test runs
Track		Queries	Same as in Team Explorer; it enables you to execute stored work item queries or create new queries
		Assign Build	Enables a tester to assign an automated build to the Test Plan
		Recommended Tests	Shows the list of all tests that have been impacted by a code change
		Project Portal	Provides a quick link to the project portal (opens a web browser)
Organize		Test Plan Manager	Lists all the Test Plans in the current Team Project
		Test Configuration Manager	Lists all test configurations
		Test Case Manager	Lists all Test Cases in the current Team Project
		Shared Steps Manager	Lists all the shared steps (reusable test steps) in the current Team Project
Lab	Lab	Environments	Contains all the physical and virtual environments ready for testing purposes

Center	Tab	Page	Description
	Test Settings	Test Settings Manager	Contains all manual and automated test settings
	Library	Environments	Lists all the environments prepped for use in testing, including environments that have been deployed
		Virtual Machines and Templates	Contains all the virtual machines available to be composed into a test environment
	Controllers	Test Controller Manager	Contains a list of all test controllers and all agents associated with those controllers
Tools	Documentation	Test Plan Summary	Generates a document with the selected Test Plans, associated Test Suites, Test Cases, Test Steps and related work items
		Test Run Summary	Generates a document with the results of the selected test runs

TEST SCRIBE AND THE TOOLS CENTER

The Tools Center does not exist when you first install MTM. After the release of Visual Studio 2010, Microsoft released a Test Scribe tool (available at <http://visualstudiogallery.msdn.microsoft.com/en-us/e79e4a0f-f670-47c2-9b8a-3b6f664bf4ae>.) (Or you can Bing "Test Scribe Visual Studio Gallery," and this link will be the first one.)

This addition is critically important to most organizations and should be installed immediately after installing MTM. The documentation it generates can be provided to users or external testers and serves as an excellent, detailed document showing the tests and test runs.

When you first start MTM, you will be asked to connect to a server (Figure 3-3), select a Team Project (Figure 3-4), and then select a Test Plan (Figure 3-5).

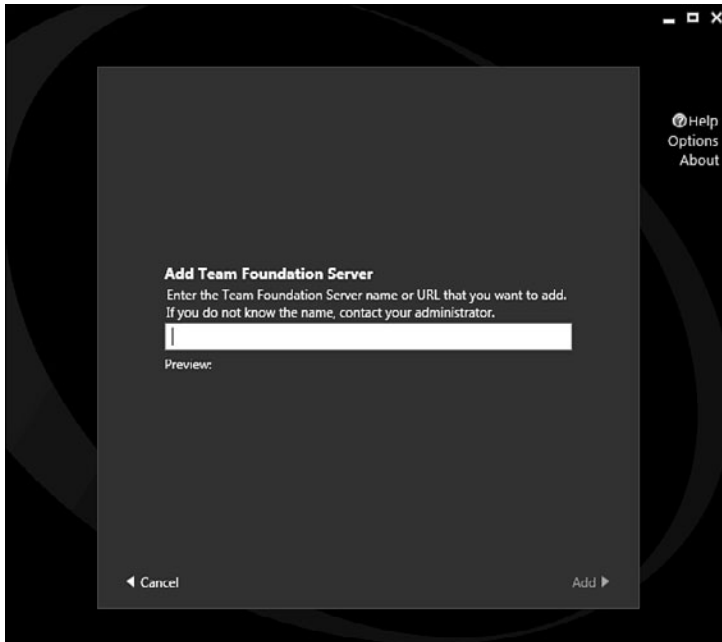


FIGURE 3-3: Connect to a Team Foundation Server

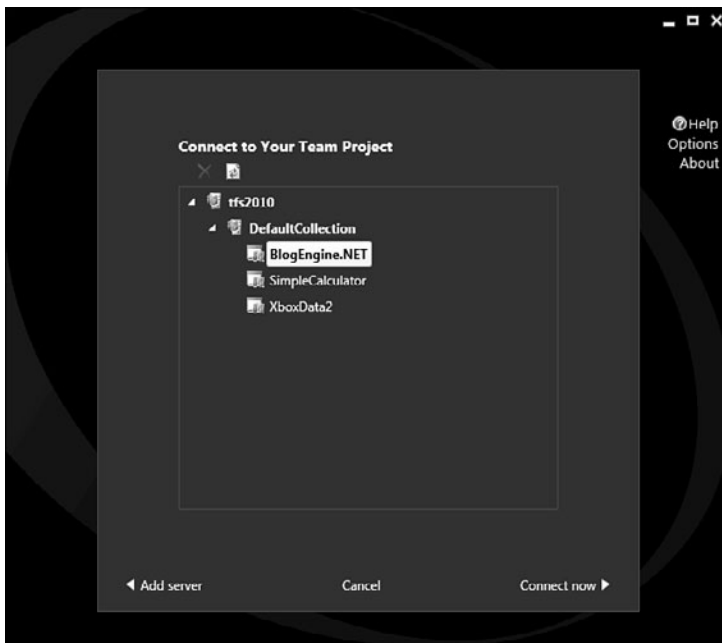


FIGURE 3-4: Connect to Your Team Project

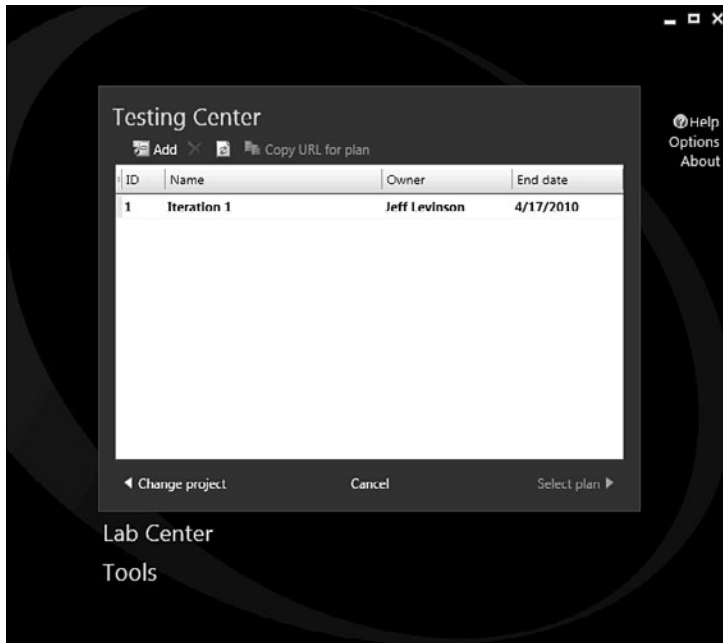


FIGURE 3-5: Select or add a Test Plan

Note the Copy URL for the plan option in Figure 3-5. MTM enables you to provide URLs to specific plans, so you can send the URL to someone who can then click it and have MTM open to the right plan. Only Active plans show up in this dialog. You can view all plans (Active and Closed) from the Testing Center, Organize Tab, Test Plan Manager page.

MTM enables you to work in one Team Project and only one Plan in that Team Project at a time, although you can change plans and projects as needed. After doing this the first time, MTM remembers your last selection, so MTM can open to the last selected Plan.

Before starting the exercises, see the section "About the Application Used in This Book" in the front matter. These exercises assume that you have followed the steps in that section.

Test Plans

Before using the testing tools, you need to understand where all the various artifacts fit together because it matters when you start to manage an actual project. Figure 3-6 shows a container view of the artifacts.

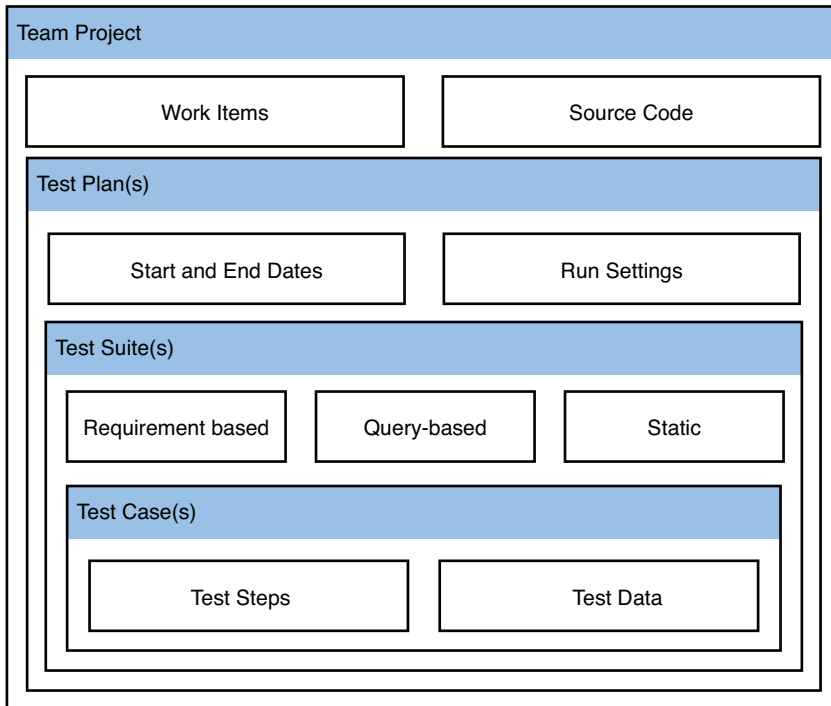


FIGURE 3-6: Relationships between Team Projects, Test Plans, Test Suites, and Test Cases

Figure 3-6 shows that a Test Plan in MTM is associated with a specific Team Project. A Test Plan is composed of one or more Test Suites, and each Test Suite is composed of one or more Test Cases. This is a straightforward structure that enables flexible reporting and easy management of the Test Plans.

EXERCISE 3-1

Create a New Test Plan

This step assumes that you have not used MTM before. If you have, but you want to work through this exercise, you need to select the Home button in the upper-left corner of the screen and select Change Project:

1. Open MTM.
2. Select Add Server, or select an existing server if the correct server is listed.
3. Select the BlogEngine.NET project, and click Connect Now.
4. On the Testing Center screen, click Add to create a new Test Plan.
5. Enter the name as **Iteration 1** and click Add.
6. Highlight the Plan, and click Select Plan.

Figure 3-7 shows the Iteration 1 Test Plan.

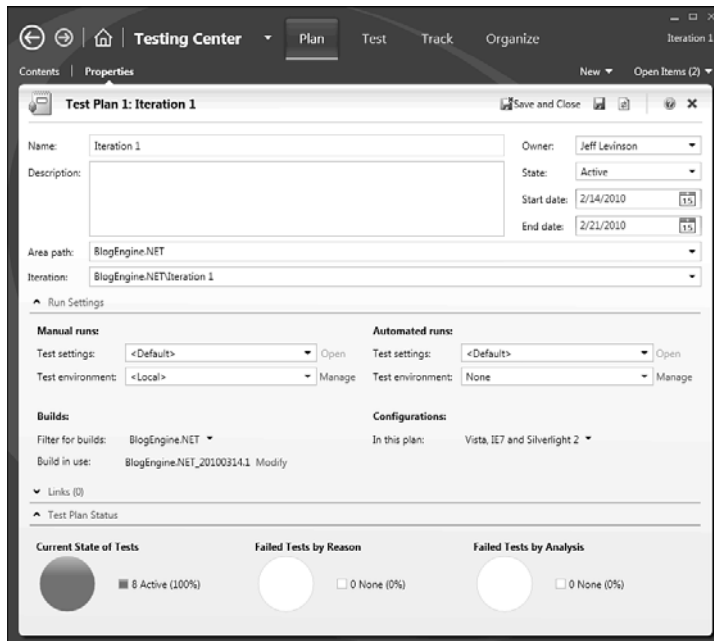


FIGURE 3-7: Test Plan

Properties

Test Plans have a name and a description, and if you use multiple Test Plans concurrently, you need to give them a descriptive name and also a more detailed description. The owner is usually the test manager but can also be a test lead if a lead is responsible for the testing that occurs within a plan. The state can either be Active or Inactive depending on whether it is currently used, and this is not customizable. Inactive Test Plans can either be previously completed Test Plans or Test Plans that have yet to be started and are still being created. The default state for new Test Plans is Active, but you might want to set the plan to Inactive if it is still being designed.

The area and iteration are the standard work item classification scheme. In general Test Plans should be related to iterations in some way (or whatever scheme the development team uses to produce software) because the testing follows the requirements or the coding, which are distinct phases in any methodology whether they are called out.

Test Plans are not work items such as a requirement, user story, or task. They are independent of the work item system. This is both a benefit and a disadvantage. The benefits are in the flexibility: the Test Plan contains more information and is more dynamic than a work item. On the other hand, items such as the Start and End date cannot be reported through a simple mechanism. You need to use the data warehouse (refer to Chapter 9, “Reporting and Metrics”) to report on Test Plans.

Run Settings

Run settings define where tests execute and what diagnostic data adapters are implemented. Figure 3-7 shows the two categories of Run settings: Manual and Automated. Manual Run settings relate to any tests executed with the Test Runner (refer to Chapter 4, “Executing Manual Tests”). Automated Run settings relate to the execution of any automated tests (refer to Chapter 6, “Automating Test Cases”) through MTM.

CHANGE THE TEST SETTINGS IMMEDIATELY

When the test settings are set to <Default> you have no control over them. You cannot set any diagnostic data adapters to run specifically or any other options associated with manual or automated runs. For the manual settings, simply select the drop-down list, and pick Local Test Run, or create a new test setting and change the properties as needed.

To create a new Run setting, go to the Lab Center, Test Settings tab, Test Settings Manager page, and copy an existing setting or add a new setting. These can then be assigned in the Test Plan Properties page. Figure 3-8 shows the Test Settings creation screen.

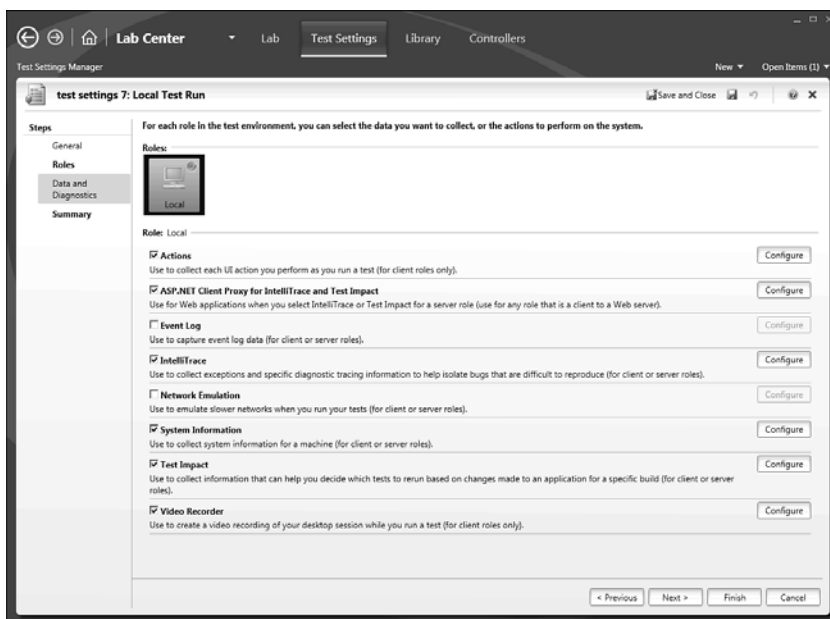


FIGURE 3-8: Test settings

Depending on whether you create an automated or manual setting, the options will be slightly different. Figure 3-8 shows a manual test setting on the Data and Diagnostics tab that contains the diagnostic data adapters. Table 3-2 lists the default diagnostic data adapters you can choose.

TABLE 3-2: Default Diagnostic Data Adapters

Collector	Description
Action Recording and Action Log	Records each step that the tester takes in the application during a manual test run.
ASP.NET Client Proxy for IntelliTrace and Test Impact	Enables you to capture IntelliTrace and Test Impact information during a test execution of an ASP.NET application. Note: This setting does not actually perform the capture; you must check the IntelliTrace and/or Test Impact collectors in addition to this collector.
Event Log	Captures selected events written to the Event Log during a test run.
IntelliTrace	Enables capturing of the debug log.
Network Emulation	Throttles the network performance based on the specified settings.
System Information	Captures system configuration information for the system on which the test is performed.
Test Impact	Records Test Impact information for calculating Test Cases affected by modified code.
Video Recorder	Records a video of all actions taken on the screen during a test run.

Diagnostic data adapters enable the test infrastructure to gather data—any particular piece of data you want. They are fully extensible and easy to create and modify (literally 20 lines of code plus whatever code is needed to collect data).

Builds

If you aren't using automated builds right now, you should be. Automated builds are one of the most effective ways to reduce the amount of time it takes to find and fix bugs. These automated builds can be Continuous Integration builds (the process of running a build immediately upon check-in to determine if the check-in broke anything) or nightly builds, and they can discover

build breaks faster and with fewer lines of code to review to find the problem. They are also critical to manual testing; although not required for automated testing, they will certainly make things easier.

Builds enable you to specify which build you can execute the tests against. After you select a build to execute the Test Cases against, MTM provides you with information related to the build. Automated builds help light up the Test Impact Analysis results and provide the testing team with a list of all changes made to the code since the build they were previously using.

The build filter enables you to filter by build definition and build quality. Chapter 5, “Resolving Bugs,” discusses build quality.

Configurations

On one hand configurations play an important part in test execution, and on the other hand they provide only metadata. Configurations enable you to specify various pieces of information about the tests you execute in the Test Plan. They also have a material effect on the number of tests that you need to execute and how you plan your Test Suites. For example, the default setting in MTM is Windows 7 and IE 8. If you have a Test Suite with 20 Test Cases, you need to execute 20 Test Cases. For every configuration that you add to a suite, all the tests need to be executed against the additional configurations as well. (By default, but you can change this.) So, if you have three configurations that you need to test against, you need to run 60 tests. The effect of configuration on testing and reporting are discussed in the “Assigning Test Configurations” section later in this chapter.

Obviously, you do not have to execute any Test Cases you don’t want to, and in many cases you can’t execute every Test Case because of the time available to you.

The “Test Configurations” section covers Test Configuration details.

Test Plan Status

This section provides status on the current Test Plan. The first pie chart lists the total number of tests broken down by successful tests, failed tests, and tests that have not yet been executed. The Failures by Type pie chart breaks down the categories of each failure. Table 3-3 shows the available categories.

TABLE 3-3: Failure Categories

Category	Description
None	Use if the test failure is a nonissue.
Regression	Where the previous test results indicate a pass.
New issue	Has not been seen before.
Known issue	Possibly because a previous run found this bug or the development team has notified the testing team that the build is ready to test, but it knows about this particular failure.
Unknown	An error occurred, but the tester is not sure what the classification of the issue is. A test lead or manager should look further at Unknown issues.

You can also provide a category for a failure type before or after it has been fixed, but leave this empty until the defect has been fixed. Table 3-4 lists the analysis categories.

TABLE 3-4: Analysis Categories (Also Called Resolution Type)

Category	Description
None	No resolution at this time.
Needs investigation	The test team has decided to do a further investigation because it isn't sure of the cause.
Test issue	Usually set if the Test Case were at fault or the setup for the test were incorrect. This might be cause for concern because if a Test Case is wrong, the requirement it is based on might also have potential inaccuracies that need to be investigated.
Product issue	A valid failure occurred in the code.
Configuration issue	Usually a failure in the configuration files or on the machine on which the test was deployed.

FAILURE AND RESOLUTION EXTENSIBILITY

You can customize the Resolution type through the process template or the object model; however, you cannot customize the Failure type. (It looks like you can do it by editing the process template, but it does not actually work because of technical reasons.)

These graphs are updated as changes are made to the Test Plan and as test runs are completed and analyzed. (For performance reasons you might need to click the Refresh button to see the latest data.) This is a great view that quickly enables a testing team to see the progress of its testing within a given plan (as shown at the bottom of Figure 3-7).

Contents

The Contents portion of a Test Plan contains information on what will be tested; that is, it contains a list of all the Test Cases broken down into *Test Suites*. Figure 3-9 shows the Contents page of the Plan tab.

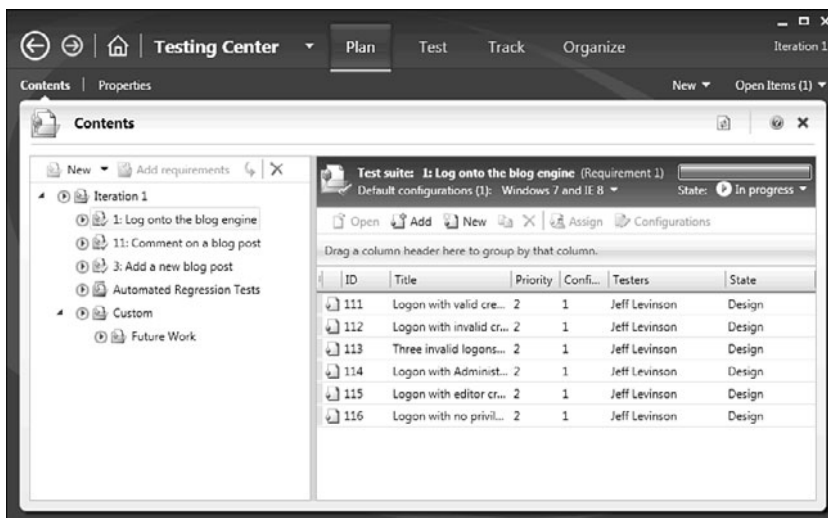


FIGURE 3-9: Test Plan contents

Refer to Figure 3-3 for the relationships between items. Test Suites can be composed in three ways: requirement-based, query-based, or customized with a static suite, and there are good uses for each of the three. The type of Test Suite is differentiated by icons next to the suite name (see Figure 3-10).

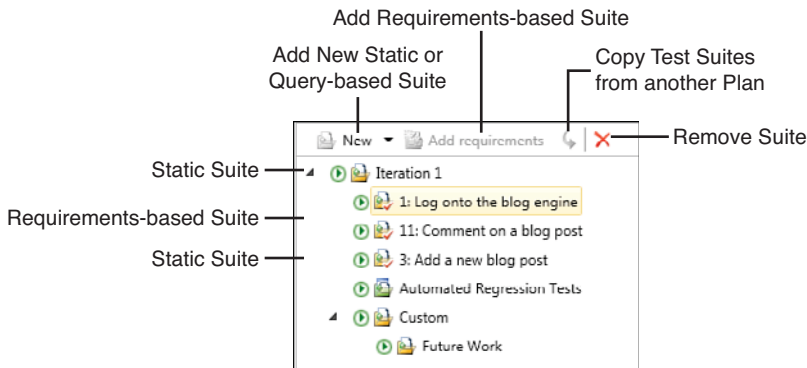


FIGURE 3-10: Test Suites

Requirements-Based Suites

For most teams developing line-of-business applications, the entire application is based around completing requirements; therefore, it makes sense that testers should test in relationship to the requirements that the developers finish. In other words, testers can rarely perform testing on partially completed requirements. They also can't perform testing on random pieces of the application because, in general, functional and integration testing relies on complete features. Even performing boundary tests must be done in the context of a requirement.

And, for the most part, customers want to know the status of their requirements. Are they close to completion? Did they pass their tests? How many bugs does a given requirement have? This is true regardless of what type of methodology you use. Grouping suites by requirement makes it extremely easy to report this information back to the customer.

To create requirements-based suites, simply select a static suite (the root node or another static suite) and click Add Requirements; then choose one or more requirements. Each requirement becomes its own suite. Any Test Cases already associated with the requirement are automatically added to the suite.

REQUIREMENTS AND WORK ITEM TYPES

Whether you use the MSF for Agile or CMMI templates, you have a requirement work item type. For the CMMI template, it is a Requirement, and for the Agile template it is a User Story. What determines a requirement from the perspective of a requirements-based suite is the category that the requirement is in. Categories are new to TFS 2010 and are a classification scheme for work item types. MTM operates on the requirement, Test Case, and bug categories. The reason it operates on categories is so that you can create a custom work item type, for example, called a Use Case that also appears in MTM if it is in the requirement category. In addition, you can create a Defect work item type that generates when you file a bug.

Query-Based Suites

These are suites created based on the results of a work item query. An example of why you might want to create a suite of this type is the need to test a specific area of your application that might be involved in different functionality. Using the requirement-based suite, you could not do this. Another reason for this type of suite can be the need to test all the bug fixes regardless of what requirement they are related to. The query-based suite simply provides you with more flexibility in selecting what you test and also enables you to run Test Cases from multiple Team Projects or requirements at the same time.

When creating this type of suite, you are limited to the results of the query, and the query specifies that you can query only work items in the Test Case category. So a query-based suite is specific to Test Cases. Because this type of suite is based on the results of a query, if the results of that query change, so will your Test Suite. Use this suite for short-term suites or suites where you don't mind them changing. An example of where this is effective is automated regression testing. You can create a query where Automation Status = Yes; when you execute the suite, all the automated tests execute.

Static Suites

A static suite is a fully custom suite; you provide the title of the suite and then add Test Cases as needed. One benefit of a static suite is that you can nest suites. This is not possible with the other two suite types. The reasons to use this type of suite can vary; however, an example of this might include final application testing where you might have time to only test requirements from various areas and iterations, and you want to break those up into subsuites so that you can roll the results up. In MTM when you select the New drop-down to add a new suite, the only two options you see are Suite and Query-Based Suite. The Suite option is the static suite.

Adding Suites and Test Cases to Your Plan

The mechanics of using the Contents window are fairly straightforward but offer a lot of options to help you control what happens when testers begin testing. The list of Test Suites is on the left side. Figure 3-6 shows a series of Test Suites starting with the Root Test Suite that is always the name of the Test Plan (Iteration 1 here). The Root Test Suite is a static suite, so you can add Test Cases directly to the root. Icons that have a red check on them are requirements-based suites. Another way to know this is to look above the list of Test Cases in the right pane; you can click the Requirement 1 link to open the requirement that these Test Cases relate to.

The Automated Regression Tests Suite in Figure 3-6 is a query-based suite, which you can tell by looking at the icon. The last suite listed, Custom, is a static suite with a Future Work subsuite that enables you to easily compose and manage your Test Suites.

You can change the default configuration for all the Test Cases here, or you can change the configuration for only individual tests. (This is not recommended because it can be difficult to keep track of which test is supposed to be run on which configuration.) You can change who the Test Cases are assigned to—either individually by selecting a Test Case and clicking the Assign button or by right-clicking the Test Suite on the left and selecting Assign Testers for All Tests (or any combination of testers to Test Cases).

In addition notice where it says State: In Progress in the upper-right corner. You can set the state to be one of three states: In Planning, In Progress,

or Completed. In Progress is the default, and tests in a Test Suite that is In Progress may be executed. A Test Suite that is In Planning will not show up on the Test tab, so those tests cannot be executed. The same is also true for Completed suites.

You can also change the columns displayed for the Test Cases by right-clicking the column headers. You can filter certain columns (any column with a discrete list) to limit what displays. (For example, you can filter the Priority column in the default list of columns.)

Finally, you have the option to open a Test Case that has been added to a suite, add Test Cases that already exist in the suite, or create new Test Cases from within MTM. Any Test Cases you create or add are automatically linked with the requirement (or user story) if the suite is a requirements-based suite with a Tested By link type. The opposite is also true; if you remove a Test Case from a requirements-based suite, the Test Case is no longer in a relationship with the requirement. (The Tests/Tested By link is deleted, but the Test Case is not deleted.)

EXERCISE 3-2

Create a Test Suite

This exercise assumes that you have completed Exercise 3-1.

1. Open MTM, if it's not already open.
2. Select Testing Center, Test Plan, Contents tab.
3. Select the Iteration 1 suite, which is the root suite and the only one that exists at this point.
4. Click Add Requirements from the toolbar for the suite name.
5. In the Add Existing Requirements to This Plan page, click Run (see Figure 3-11).
6. Select the requirement As the Blog Author I Want to be Able to Log onto the Blog Engine, and click Add Requirements to Plan in the lower-right corner.

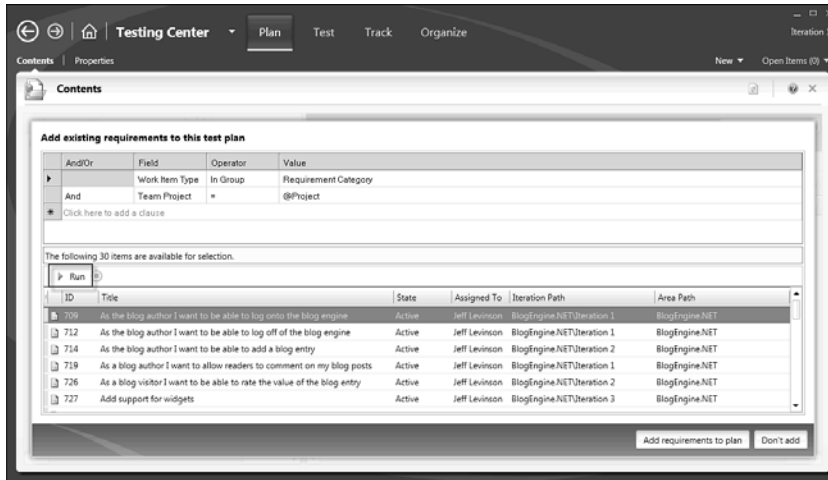


FIGURE 3-11: Add Existing Requirements to This Test Plan page

Testing Configurations

Testing configurations are configurable and can have an impact on the number of tests that need to be executed (mentioned previously). Test configurations specify any particular piece of information needed to ensure that your software is tested against all possible configuration options users could have on their machine.

As of this release, test configurations are strictly metadata. That is, they do not have any impact on the test runs and cannot be used to specify the hardware or software a particular test is actually executed against.

The most typical example is using different browsers to ensure the rendering works correctly. Added to that may be the operating system those browsers run on. The two default configuration options are Operating System and Browser; to this you can add other things such as a Silverlight version or a particular piece of hardware, such as a webcam.

The biggest benefit to using test configurations is reporting results. All your test results can be broken down into configurations. In addition you have to write the Test Cases only one time, but this presents other issues, such as that the actions you take on one configuration may not be valid on another configuration. In some cases the differences may be so great it doesn't make sense to use the same Test Case. Consider these items when deciding on how to use test configurations.

Managing Test Configurations

You can access the Test Configuration Manager in two ways. The first is to go to Testing Center, Plan, Properties and select the drop-down arrow next to the configuration; then click Manage. The easier way is to go to Testing Center, Organize, Test Configuration Manager. This brings up the screen shown in Figure 3-12.



FIGURE 3-12: Test Configuration Manager

The Manage Configuration Variables option enables you to create new configuration categories. You can also add new values to an existing configuration variable.

EXERCISE 3-3**Adding a New Configuration Variable**

To add a new configuration variable, follow these steps:

1. Click Manage Configuration Variables.
2. Click New Configuration Variable.
3. Enter **Silverlight Version** for the name.
4. Enter **Default Silverlight Versions** for the description.
5. In Allowed Values, enter the following (shown in Figure 3-13): **1, 2, 3,** and **4.**
6. Click Save Configuration Variables.

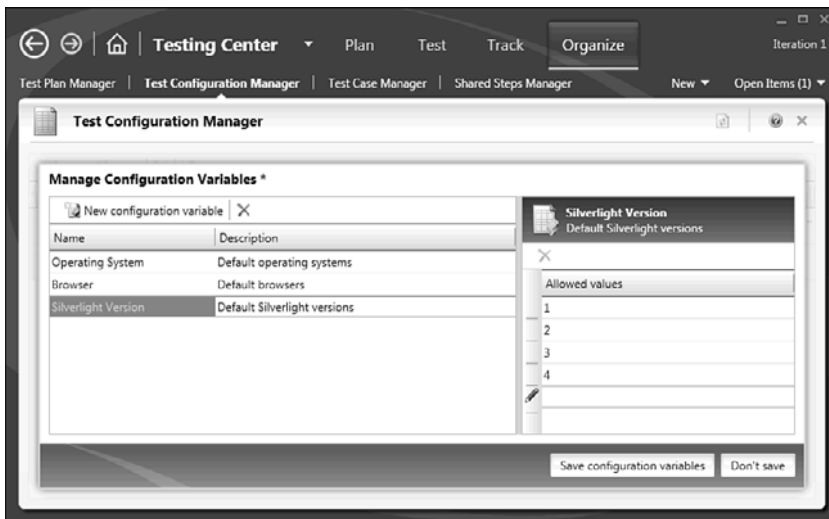


FIGURE 3-13: Silverlight Version Configuration Variable

The variables themselves cannot be used directly. You need to create an actual configuration composed of one or more configuration variables.



Index

A

- Acceptance Test Driven Development (ATDD), 26
- acceptance testing, 25
 - ATDD (Acceptance Test Driven Development), 26
- access to Test Cases, 6
- active state, 93
- adding
 - recorded steps, 164-165
 - validations, 157-164
- advantages of Microsoft Visual Studio 2010, 5
- automated tests, 9-10
- communication, 5-6
- development and testing process flow, 7-9
- metrics, 10-12
- project visibility, 6
- agents, running as interactive processes, 185
- Agile, updating bugs, 114
- agile practices, 23
- agile testing, 20
- Agile Testing: A Practical Guide for Testers and Agile Teams (Crispin and Gregory), 20
- ALM (Application Lifecycle Management), 19
- analysis categories, 42-43
- analysis phase (Test Cases), 56-61
- Analysis section, detailed test results, 95
- Application Lifecycle Management (ALM), 19
- applications
 - BlogEngine.NET. *See* application, xxiv
- Asimov, Isaac, 269
- ASP.NET, MSAA, 162
- assigning
 - builds, 127-129
 - test configurations, 51-53
 - testers, 53-54
- Associated Change sets, 124
- Associated Work Items, 124
- associating
 - Coded UI Tests and Test Cases, 178-181
 - Unit Tests, 181
- ATDD (Acceptance Test Driven Development), 26

- Attachments section, detailed test results, 98-100
- attributes, 186
- automated builds, creating, 191
- automated builds (Test Plans), 40-41
- automated test settings, 221-222
 - Lab Management workflow, 222-231
- automated testing, 24
- automated testing framework, 139-141
- automated tests, 9-10
 - creating from manual tests, 141-142
 - coded UI tests, 144-157
 - examining generated web application coded UI tests, 142-144
 - executing, 183-184
 - from command line, 190
 - local execution, 184
 - local execution with remote collection, 184
 - in MTM, 191-196
 - remote execution, 185-189
 - executing with Team Build, 202-203
 - issues with, 205
 - custom dialogs, 205-207
 - running through MTM, 233-234
- automating manual Test Cases, 142
- automation, choosing to automate, 136-138
- B**
- best practices for parameterized tests, 88
- binaries, 82
- black-box testing, 21
- Blocked field, 113
- blocked test cases, 101
- BlogEngine.NET application, xxiv
- BlogEntryHTMLBasicTestCodedUITestMethods, 167
- boundary cases, 21
- Browser Window class, 150
- bug count per feature, 11, 279
- bug count per phase, 11, 278
- bug reactivations, 276
 - comparing measurements, 277
 - lowering, 277
 - measuring, 276
- Bug Status reports, 244-245
- Bug Trends reports, 245-246
- Bug work item type, 107-110
 - customer reported bugs, 110
 - reactivations, 111
 - test team reported bugs, 110
 - triaging bugs, 110
- Bug work item type, generated bugs, 116-119
- bug workflow, 113
- bugs
 - bug count per feature, 11
 - bug count per phase, 11
 - differences and modifications, 112-116
 - finding and filing, 88-89
 - fixing, 122-124
 - Associated Change sets, 124
 - Associated Work Items, 124
 - impacted tests, 125
 - regression bugs, 11, 16, 138
 - total bug count, 11
 - triaging, 116
 - updating in Agile, 114
 - verifying fixes, 129-131
- \$(Build Location), 228
- Build Quality Indicators reports, 246-248

- build reports, 232
- Build Result Count Trend, 257
- Build Success over Time reports, 248-249
- Build Summary reports, 249-250
- build warnings, 204
- building quality at the beginning of projects, 17
- builds, 82
 - assigning, 127-129
 - automated builds, creating, 191
 - dual purpose, 233
 - lab builds, executing, 231-232
 - quality, 125-127
 - retention, 130
 - work items, 125
- builds (Test Plans), 40-41
- built-in reports, 242-244
 - Bug Status, 244-245
 - Bug Trends, 245-246
 - Build Quality Indicators, 246-248
 - Build Success over Time, 248-249
 - Build Summary, 249-250
 - reactivations, 246
 - Stories Overview, 250-251
 - Test Case Readiness, 251-252
 - Test Plan Progress, 252-253
- built-in templates, 224
- business value of software quality, 14

C

- capturing metrics, 272
- challenges of software testing, 1-3
- Cigna Corporation, 4
- cleaning up tests, 207
- closing IE Browser window, 165
- Code Complete (McConnell), 13

- code coverage, 11
- Coded UI Test builds, 159
- Coded UI Tests, 24, 144-147
 - associating with Test Cases, 178-181
 - maintaining, 154
 - recording from scratch, 165
 - running through MTM, 196-199
 - searching for controls, 148-157
- CodedUITestMethods, 170, 175
- combining tests, 178
- command line, executing automated tests, 190
- communication, improving, 5-6
- \$(ComputerName_), 228
- Computer Science Corporation, 4
- configurations (Test Plans), 41
- configuring virtual environments, 217-218
- connecting to Team Foundation Server, 33-34
- construction phase (Test Cases), 61-62
- Contents section (Test Plans), 43
 - query-based suites, 45
 - requirements-based suites, 44-45
 - static suites, 46
- Continuous Integration builds, 40
- Control Specific section, validations, 161
- controls, searching for (Coded UI Tests), 148-157
- corner cases, 21
- cost of poor software quality, 3-5
- costs, defect cost, 11
- Covey, Stephen R., 26
- Crispin, Lisa, 20
- cube (SSAS), 240-242
- Cunningham, Ward, 246
- custom dialogs, automated tests, 205-207

- customer reported bugs, 110
- customizing
 - process templates, 115, 270
 - work items, 61

D

- dashboards, 254
- data, gathering diagnostic data, 235
- data driven test cases, 77
- data sources, Test Cases, 168
- database unit testing, 22
- default diagnostic data adapters (Test Plans), 40
- defect cost, 11
- defect root cause, 11
- Deploy.cmd, 227
- DeployDatabase.cmd, 229
- deployed products, testing, 27-28
- deployed VMs, 214
- deploying test code, 127
- detailed test results, 95
 - Analysis section, 95
 - Attachments section, 98-100
 - Links section, 100
 - Result History section, 100-101
 - Test Step Details section, 96-97
- developer-focused testing, 184
- developers, testing, 136
- development
 - ATDD (Acceptance Test Driven Development), 26
 - FDD (feature-driven development), 65-66
 - moving from one iteration to another, 67-68
- development of Lab Management, xviii-xix

- development of Microsoft Visual Studio Test Professional 2010, xvii-xix
- diagnostic data, gathering, 235
- diagnostic data adapters (Test Plans), 40
- differences, bugs, 112-116
- documentation, MSDN, 253
- done, definition of, 18
- dual purpose builds, 233
- dynamic values, 172-178

E

- edge cases, 21
- editing test steps, 73
- encrypted passwords, 148
- end of projects, building quality at, 17
- environments, setting up (executing automated tests), 193-196
- examining test results, 92-93
 - detailed test results, 95-101
 - test run results, 93-94
- Excel Services, 243
- Excel Services reports, 253-254
- Exception Data, 120
- executing
 - automated tests, 183-184
 - from the command line, 190
 - in MTM, 191-196
 - local execution, 184
 - local execution with remote collection, 184
 - remote execution, 185-189
 - with Team Build, 202-203
- lab builds, 231-232
- tests, 85-86, 159
 - parameterized tests, 87
- expectations of software quality, 15

exploratory testing, 23
MTM, 101-104
external software quality, 13-14

F

failures, failure categories, 42
FBI's Virtual Case File system, 4
FDD (feature-driven development), 65-66
feature-driven development (FDD), 65-66
filing bugs, 88-89
finding bugs, 88-89
first-time defect rate, 273
causes of, 273
comparing measurements, 275
lowering, 274-275
measuring, 273-274
related metrics, 276
fixing bugs, 122-124
Associated Change sets, 124
Associated Work Items, 124
impacted tests, 125
formal reviews, reducing general bug count, 280
Found in Environment field, 114
FQDN (fully qualified domain name), 228
frameworks, automated testing, 139-141
functional testing, 24

G

general bug counts, 277
measuring, 278-279
reducing, 279-282
Generate Code dialog, 164
generated bugs, 116-119

generated control class, 174
generated web application coded UI tests, 142-144
generating
reports from work item queries, 255-256
ValidateHTMLInfo code, 171
goals of software testing, 19
gray-box testing, 22
Gregory, Janet, 20
Group Policy Editor, 206

H

Heinlein, Robert, 269
How Found field, 114

I

IE Browser windows, closing, 165
IE DOM (Internet Explorer Document Object Model), 136
IEFrame properties, 162
impacted tests, 125, 131-132
importing
Test Cases, 77
VMs, 210-212
improving communication, 5-6
inconsistency issues, parameterized Coded UI Tests, 168-169
resolving, 169-170
increasing project visibility, 6
initial design (Test Cases), 56-61
integration testing, 23-24
IntelliTrace, 119-122
IntelliTrace Settings, 80
internal software quality, 13-14
\$(InternalComputerName_ring, 228



Internet Explorer Document Object Model (IE DOM), 136
iterations, moving from one iteration to another, 67-68

K

Kelvin, Lord, 269
Kristensen, Mads, xxiv

L

lab builds, executing, 231-232
Lab Management, 194, 209
 development of, xviii-xix
Lab Management workflow, 222-231
Links section, detailed test results, 100
load testing, 24
local execution, automated testing, 184
 with remote collection, 184
LoggedOnUserPreFilledTestClass,
 142-143
lowering
 bug reactivations, 277
 defect rates, 274-275

M

macros, 227-228
maintainability, 16
management, test management, 27
manual black-box testing, xvii-xviii
manual Test Cases, creating, 74-75
manual tests in virtual environments,
 234-238
McConnell, Steve, 13
mean time between failures (MTBF), 16
measuring
 bug reactivations, 276
 defect rates, 273-274
 general bug counts, 278-279

metrics, 268-271
 bug reactivations, 276-277
 capturing, 272
 explained, 10-12
 first-time defect rate, 273
 causes of, 273
 comparing measurements, 275
 lowering, 274-275
 measuring, 273-274
 related metrics, 276
 general bug counts, 277
 measuring, 278-279
 reducing, 279-282
 what to measure, 271-272
Microsoft Active Accessibility. *See* MSAA
Microsoft Environment Viewer, 218
Microsoft Excel
 fields and placement, 267
 reporting with, 254
 creating generated reports, 255
 testing measures, 256-257
 reports, creating Test Cases, 257-268
Microsoft Team Foundation Server 2010,
 Lab Management, 194, 209
Microsoft Test Manager. *See* MTM
Microsoft Visual Studio Test Professional
 2010, development of, xvii, xix
middle of project, building quality at, 17
modifications, bugs, 112-116
MOSS (Microsoft Office SharePoint
 Server), 239
MSAA (Microsoft Active
 Accessibility), 136
 ASP.NET, 162
MSBuild, 8
MSDN (Microsoft Developer Network),
 documentation, 253
MSF for Agile Bug work item types, 108
MSF for CMMI Bug work item type, 109

MSI packages, 217
 MSTest.exe, 190
 MTBF (mean time between failures), 16
 MTM (Microsoft Test Manager), 8
 connecting to Team Foundation Server,
 33-34
 executing automated tests, 191-192
 setting up physical environment,
 193-196
 explained, 30
 exploratory testing, 101-104
 managing virtual environments,
 210-216
 navigation controls, 30
 navigation layout, 31
 running automated tests, 233-234
 running Coded UI Tests, 196-199
 creating test settings, 199-201
 selecting team projects, 34
 table of components, 31-33
 Test Plans. *See* Test Plans
 Tool Center, 33

N

NameoftheblogShortdeDocument, 163
 naming test assemblies, 192
 navigation controls (MTM), 30
 navigation layout (MTM), 31
 need for testers, 3-5
 nightly builds, 40
 nonfunctional requirements
 explained, 15
 maintainability, 16
 reliability, 16
 security, 16
 usability, 16
 Nyveldt, Al, xxiv

O

Original Estimate field, 114

P

parameterized Coded UI Tests, 166-168
 inconsistency issues, 168-169
 resolving inconsistency issues, 169-170
 parameterized test cases, creating, 78
 parameterized tests
 best practices, 88
 executing, 87
 passwords, encrypted, 148
 pausing test runs, 89-90
 PeopleSoft, 4
 physical environments, 198
 physical machines, 194
 PivotTable field sections, 261
 plans. *See* test plans
 Point Count Trend, 257
 poor software quality, cost of, 3-5
 PowerShell, 227
 pre-user acceptance testing, 25
 process, impact on quality, 19
 process flow, 7-9
 process templates, customizing, 115, 270
 project visibility, increasing, 6
 projects
 relationship with test suites, test cases,
 and Test Plans, 36
 selecting in MTM (Microsoft Test
 Manager), 34
 properties of Test Plans, 38
 Proposed Fix field, 115
 purpose of software testing, 19

Q

quality

- as team effort, 18
 - building at beginning of project, 17
 - builds, 125-127
 - business value, 14
 - cost of poor software quality, 3-5
 - definition of done, 18
 - expectations, 15
 - impact of process on, 19
 - internal versus external, 13-14
 - maintainability, 16
 - reliability, 16
 - requirements, 14
 - security, 16
 - usability, 16
- query-based suites, 45

R

- Range Selector, 102
- reactivations, 111, 246
- recorded steps, adding, 164-165
- recording Coded UI Tests from scratch, 165
- reducing general bug counts, 279-281
 - test normal pathfirst, 281-282
- regression bugs, 11, 16, 138, 279
- regression testing, 25
- related metrics, defect rates, 276
- reliability, 16
- remote collection, local execution (automated tests), 184
- remote execution, automated testing, 185-189
- replaying test steps, 90-91
- reporting with Microsoft Excel, 254
 - creating generated reports, 255
 - testing measures, 256-257
- reporting structures, 240-242
- reports
 - built-in reports, 242-244
 - Bug Status, 244-245
 - Bug Trends, 245-246
 - Build Quality Indicators, 246-248
 - Build Success over Time, 248-249
 - Build Summary, 249-250
 - reactivations, 246
 - Stories Overview, 250-251
 - Test Case Readiness, 251-252
 - Test Plan Progress, 252-253
 - Excel Services, 253-254
 - generating from work item queries, 255-256
 - User Stories, creating Test Cases, 257-268
- requirements for software quality, 14
- requirements coverage, unit tests, 141
- requirements-based suites, 44-45
- resolution types, 42-43
- resolving data inconsistency,
 - parameterized Coded UI Tests, 169-170
- Result Count, 257
- Result Count Trend, 257
- Result History section, detailed test results, 100-101
- resuming test runs, 89-90
- Ritchie, Arthur David, 269
- Root Cause field, 115
- Run settings (Test Plans), 38-40
- Run Tests page, 92

running
 automated tests through MTM, 233-234
 Coded UI Tests through MTM, 196-199
 tests, 79-80
 Test Runner, 80-84

S

SAP, 3
 Scheduling Test Cases, 64-65
 Science Applications International Corporation, Virtual Case File system, 4
 SCRUM, 20
 SCVMM (System Center Virtual Machine Manager), 210
 search conditions, 176
 searching for controls, coded UI tests, 148-157
 security, 16
 security groups, test controllers, 194
 server names, 233
 servers, Team Foundation Server (connecting to), 33-34
 service level agreements (SLAs), 16
 SetupWebServer.cmd, 228
 Share-Point, 243
 Shared Step, executing tests, 86
 shared steps
 creating, 76-77
 Test Case work item type, 75
 shipped products, testing, 27-28
 Siebel Systems, 4
 SLAs (service level agreements), 16
 smoke tests, 23
 snapshots, 217
 of environments, 219-221
 software quality
 as team effort, 18
 building at beginning of project, 17
 business value, 14
 definition of done, 18
 expectations, 15
 impact of process on, 19
 internal versus external, 13-14
 maintainability, 16
 reliability, 16
 requirements, 14
 security, 16
 usability, 16
 software testing, need for testers, 4
 speeding up testing, 234
 SSAS (SQL Server Analysis Services), 239
 cube, 240-242
 SSRS (SQL Server Reporting Services), 239
 built-in reports, 242-244
 Bug Status, 244-245
 Bug Trends, 245-246
 Build Quality Indicators, 246-248
 Build Success over Time, 248-249
 Build Summary, 249-250
 reactivations, 246
 Stories Overview, 250-251
 Test Case Readiness, 251-252
 Test Plan Progress, 252-253
 static suites, 46
 Test Cases, 80
 Stories Overview reports, 250-251
 suites. *See* Test Suites
 Symptom field, 115
 System Center Virtual Machine Manager (SCVMM), 210
 system testing, 21, 25

T

- Tcm.exe, 190
- Team Build, executing automated tests, 202-203
- Team Foundation Server. *See* TFS
 - connecting to, 33-34
- Team Project Collections. *See* TPCs
- teams, involvement in building software quality, 18
- Technical Debt, 246
- templates
 - built-in, 224
 - process templates, customizing, 115, 270
 - Test Approach Word template, 30
- test approach, 30
- Test Approach Word template, 30
- test assemblies, naming, 192
- test attachments, 119
- Test Case Readiness reports, 251-252
- Test Case work item type, 72-74
 - data driven test cases, 77
 - shared steps, 75
 - creating, 76-77
- Test Cases
 - access to, 6
 - adding to Test Plans, 46-47
 - assigning testers to, 53-54
 - associating with Coded UI Tests, 178-181
 - automating manual Test Cases, 142
 - blocked, 101
 - creating manual, 74-75
 - data sources, 168
 - FDD (feature-driven development), 65-66
 - handling different test, configurations, 68
 - importing, 77
 - moving from one iteration to another, 67-68
 - parameters, creating, 78
 - relationship with team projects, test suites, and Test Plans, 36
 - scheduling and tracking, 64-65
 - static suites, 80
 - testing workflow, 55-56
 - analysis and initial design, 56-61
 - construction, 61-62
 - user acceptance testing, 62-64
 - User Stories Report, 257-268
- test code, deploying, 127
- Test Configuration Manager
 - accessing, 49
 - adding configuration variables, 50
 - assigning test configurations, 51-53
 - creating test configurations, 51
- test configurations
 - accessing Test Configuration Manager, 49
 - adding configuration variables, 50
 - assigning, 51-53
 - benefits of, 49
 - creating, 51
 - explained, 48
 - handling different test configurations, 68
- Test Controller Configuration tool, 193
- Test Impact Analysis (TIA), 7, 125
- Test List Editor, 158
- test management, 27
- Test Manager. *See* MTM (Microsoft Test Manager)
- test parameters, 77
- Test Plan Progress reports, 252-253

- Test Plan Status section (Test Plans), 42
 - analysis categories, 42-43
 - failure categories, 42
- Test Plans, 55. *See also* testing workflow
 - builds, 40-41
 - configurations, 41
 - Contents section, 43-44. *See also* Test Suites
 - static suites, 46
 - creating, 37
 - default diagnostic data adapters, 40
 - properties, 38
 - relationship with team projects, test suites, and test cases, 36
 - Run settings, 38-40
 - Selecting, 35
 - Test Cases
 - adding to plans, 46-47
 - assigning testers to, 53-54
 - FDD (feature-driven development), 65-66
 - handling different test configurations, 68
 - moving from one iteration to another, 67-68
 - scheduling and tracking, 64-65
 - Test Plan Status section, 42
 - analysis categories, 42-43
 - failure categories, 42
 - Test Suites
 - adding to plans, 46-47
 - creating, 47
 - query-based suites, 45
 - requirements-based suites, 44
 - static suites, 46
 - test results, examining, 92-93
 - detailed test results, 95-101
 - test run results, 93-94
 - Test Results, test attachments, 119
 - test run results, 93-94
 - Test Runner, 80-84
 - bugs, finding and filing, 88-89
 - pausing and resuming test runs, 89-90
 - replaying test steps, 90-91
 - Test Runner (TR), 71
 - test runs, pausing and resuming, 89-90
 - Test Scribe tool, 33
 - test settings, creating, 199-201
 - Test Step Details section, detailed test results, 96-97
 - test steps
 - editing, 73
 - replaying, 90-91
 - Test Suites, 43-44
 - adding to Test Plans, 46-47
 - creating, 47
 - query-based suites, 45
 - relationship with team projects, test cases, and Test Plans, 36
 - requirements-based suites, 44-45
 - static suites, 46
 - test team reported bugs, 110
 - testers
 - assigning, 53-54
 - need for, 3-5
 - testing mindset, 20
 - testing
 - automated testing framework, 139-141
 - developer-focused testing, 184
 - exploratory testing with MTM, 101-104
 - manual black-box testing, xvii-xviii
 - speeding up, 234
 - testing measures, Microsoft Excel, 256-257
 - testing mindset, 20

- testing workflow, 55-56
 - analysis and initial design, 56-61
 - construction, 61-62
 - user acceptance testing, 62-64
- tests
 - automated tests. *See* automated tests
 - cleaning up, 207
 - Coded UI Tests. *See* Coded UI Tests
 - combining, 178
 - executing, 85-86, 159
 - parameterized tests, 87
 - generated web application coded UI tests, 142-144
 - impacted tests, 131-132
 - manual tests in virtual environments, 234-238
 - parameterized Coded UI Tests, 166-168
 - inconsistency issues, 168-169
 - resolving inconsistency issues, 169-170
 - parameterized tests, best practices, 88
 - running, 79-80
 - Test Runner, 80-84
 - Test Runner
 - finding and filing bugs, 88-89
 - pausing and resuming test runs, 89-90
 - replaying test steps, 90-91
 - Unit Tests
 - associating, 181
 - requirements coverage, 141
 - tests coded UI tests, 144-147
 - searching for controls, 148-157
 - TFS (Team Foundation Server), 12, 239
 - automated tests, 9-10
 - metrics, explained, 10-12
 - TIA (Test Impact Analysis), 125, 132
 - explained, 7-9

- time, 241
- Tool Center, 33
- tools, Test Controller Configuration tool, 193
- total bug count, 11
- TPCs (Team Project Collections), 240
- tracking Test Cases, 64-65
- transparency, 6
- triaging bugs, 110, 116

U

- UAT (User Acceptance Testing), 21, 63
- UI test files, 154-155
- UIA (User Interface Automation), 136
- UISigninDocument class, 150-152
- UISigninWindowsInterneWindow class, 148
- unit testing, 22
- Unit Tests, associating, 181
 - requirements coverage, 141
- updating bugs in Agile, 114
- usability, 16
- User Stories reports, Test Cases (creating), 257-268
- user acceptance testing, 62-64
- User Acceptance Testing (UAT), 21
- User Interface Automation (UIA), 136
- users, expectations of software quality, 15

V

- ValidateHTMLInfo code, generating, 171
- validations
 - adding, 157-158, 160-164
 - multiple validations, 158
- values, dynamic values, 172-178
- variables, adding to test configurations, 50

- verifying bug fixes, 129-131
- videos, 97
- Virtual Case File system (FBI), 4
- virtual environments
 - configuring, 217-218
 - managing with MTM, 210-216
 - manual tests, 234-238
 - options, 215
 - snapshots, 219-221
 - versus virtual machines, 213
- virtual machines, 194
 - versus virtual environments, 213
- virtualized testing, 90
- visibility of projects, increasing, 6
- Visual Studio Test Professional 2010,
 - development of, xvii, xviii
- VMs (virtual machines), importing,
 - 210-212
- VMWare virtual machines, 194
- vsdbcmd command-line tool, 229

W-X-Y

- Waste Management, Inc., 3
- white-box testing, 21
- Windows SharePoint Services (WSS), 243
- work item queries, generating reports,
 - 255-256
- work items
 - builds, 125
 - customizing, 61
- workflow, 55-56
 - analysis and initial design, 56-61
 - bugs, 113
 - construction, 61-62
 - user acceptance testing, 62-64
- WSS (Windows SharePoint Services), 243

Z

- zero defect releases, 3
- Zumdahl, Steven S., 269