



Professional Expertise Distilled

# Software Testing using Visual Studio 2012

Learn different testing techniques and features of Visual Studio 2012 with detailed explanations and real-time samples

Satheesh Kumar N  
Subashni S

**[PACKT]** enterprise   
PUBLISHING professional expertise distilled

# Software Testing using Visual Studio 2012

Learn different testing techniques and features of  
Visual Studio 2012 with detailed explanations and  
real-time samples

**Satheesh Kumar N**

**Subashni S**



BIRMINGHAM - MUMBAI

# Software Testing using Visual Studio 2012

Copyright © 2013 Packt Publishing

All rights reserved. No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior written permission of the publisher, except in the case of brief quotations embedded in critical articles or reviews.

Every effort has been made in the preparation of this book to ensure the accuracy of the information presented. However, the information contained in this book is sold without warranty, either express or implied. Neither the author, nor Packt Publishing, and its dealers and distributors will be held liable for any damages caused or alleged to be caused directly or indirectly by this book.

Packt Publishing has endeavored to provide trademark information about all of the companies and products mentioned in this book by the appropriate use of capitals. However, Packt Publishing cannot guarantee the accuracy of this information.

First published: December 2010

Second Edition: July 2013

Production Reference: 1190713

Published by Packt Publishing Ltd.  
Livery Place  
35 Livery Street  
Birmingham B3 2PB, UK.

ISBN 978-1-84968-954-0

[www.packtpub.com](http://www.packtpub.com)

Cover Image by Artie Ng ([artherng@yahoo.com.au](mailto:artherng@yahoo.com.au))

# Credits

**Authors**

Satheesh Kumar N  
Subashni S

**Reviewers**

Ahmed Ilyas  
Ken Tucker  
Hulot  
Kalyan

**Acquisition Editor**

Anthony Lowe

**Lead Technical Editor**

Mayur Hule

**Technical Editors**

Ruchita Bhansali  
Krishnaveni Haridas  
Pratik More  
Anita Nayak  
Larissa Pinto

**Project Coordinator**

Anugya Khurana

**Proofreader**

Dan McMahon

**Indexer**

Tejal Soni

**Production Coordinator**

Kyle Albuquerque

**Cover Work**

Kyle Albuquerque

# About the Authors

**Satheesh Kumar N** holds a Bachelor's degree in Computer Science engineering and has around 17 years of experience in managing the software development life cycle, developing live projects, and program management. He started his career by developing software applications using Borland software products. He worked for multiple organizations in India, the UAE, and the US. His main domain expertise is in retail and he is currently working in Bangalore as a Program Delivery Manager for the top retailer in UK. He is currently handling five agile scrum teams for delivering the website features. His experience also includes implementation and customization of Microsoft Dynamics for an automobile sales company in UAE. He works with the latest Microsoft technologies and has published many articles on LINQ and other features of .NET. He is a certified PMP (Project Management Professional).

He has also authored *Software Testing using Visual Studio Team System 2008* and *Software Testing using Visual Studio 2010* for Packt Publishing.

---

I would like to thank my wife for helping me in co-authoring and supporting me in all the ways to complete this book. I would also like to thank my family members and friends for their continuous support in my career and success.

---

**Subashni S** holds a Bachelor's Degree in Computer Science engineering and has around 15 years of experience in software development and testing life cycle, project, and program management. She is a certified PMP (Project Management Professional), CSTM (Certified Software Test Manager), and ITIL V3 Foundation certified. She started her career as a DBA in Oracle 8i technology, and later developed many software applications using Borland software products for a multinational company based in Chennai, and then moved to Bangalore. She is presently working for a multinational company, in the area of Project Management for developing and testing projects. She is also currently working for one of the top multinational companies headquartered at Austin, Texas.

She has also authored *Software Testing using Visual Studio Team System 2008* and *Software Testing using Visual Studio 2010* for Packt Publishing.

---

I would like to thank my husband for helping me in co-authoring and supporting me in all the ways to complete this book. I would also like to thank my other family members and friends for their continuous support in my career and success.

---

# About the Reviewers

**Ahmed Ilyas** has a BENG degree from Napier University in Edinburgh, Scotland, where he majored in software development. He has 15 years of professional experience in software development.

After leaving Microsoft, he has ventured into setting up his consultancy company offering the best possible solutions for a magnitude of industries and providing real world answers to those problems, and only uses the Microsoft stack to build these technologies and be able to bring in the best practices, patterns, and software to his client base to enable long-term stability and compliance in the ever-changing software industry. He has also tried to improve software developers around the globe, pushing the limits in technology.

This went on to being awarded three times the MVP in C# by Microsoft for “providing excellence and independent real world solutions to problems that developers face.”

With the breadth and depth of the knowledge he has obtained not only from his research, but also with the valuable wealth of information and research at Microsoft, the motivation and inspirations come from this, with 90 percent of the world using at least one form of Microsoft technology.

Ahmed Ilyas has worked for a number of clients and employers. With the great reputation that he has, this has resulted in having a large client base for his consultancy company, Sandler Ltd (UK) which includes clients from different industries, from media to medical and beyond. Some clients have included him on their “approved contractors/consultants” list which include ICS Solution Ltd and has been placed on their “DreamTeam” portal and also CODE Consulting/EPS Software ([www.codemag.com](http://www.codemag.com)) (based in USA).

Ahmed Ilyas has also been involved in the past in reviewing books for *Packt Publishing* and wish to thank them for the great opportunity once again.

---

I would like to thank the author/publisher of this book for giving me the great honor and privilege in reviewing the book. I would also like to thank my client base and especially Microsoft Corporation and my colleagues over there for enabling me to become a reputable leader as a software developer in the industry, which is my passion.

---

**Ken Tucker** is a Microsoft MVP from 2003-2013. He has also worked for Seaworld Parks and Entertainment.

---

I would like to thank my wife Alice-Marie.

---

**Carlos Hulot** has been working in the IT area for more than 20 years in different capabilities, from software development, project management to IT marketing, product development and management. Carlos has worked for multinational companies such as Royal Philips Electronics, PricewaterhouseCoopers, and Microsoft. Currently Carlos is working as an independent IT consultant. Carlos is a Computer Science lecturer in two Brazilian universities. Carlos holds a Ph.D. in Computer Science and Electronics from the University of Southampton, UK, and a B.Sc. in Physics from University of São Paulo, Brazil.

**Kalyan Bandarupalli** is currently working in Oxford University, UK. His professional career started as a software engineer and then senior software developer and software architect. He is a senior consultant, who uses Microsoft technologies to develop applications. Since 2003, he has been working as a Microsoft technology developer.

He was far more concerned about the technical implementation of software, but in the past few years focus has changed to more architectural implementation of software. He recently (June 2008) started a blog ([www.techbubbles.com](http://www.techbubbles.com)), because he wanted to share his learning experience to help other people learn about new technologies in Microsoft software. This blog helps IT professionals and developers around the world to develop applications using Microsoft technologies.

# www.PacktPub.com

## Support files, eBooks, discount offers and more

You might want to visit [www.PacktPub.com](http://www.PacktPub.com) for support files and downloads related to your book.

Did you know that Packt offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at [www.PacktPub.com](http://www.PacktPub.com) and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at [service@packtpub.com](mailto:service@packtpub.com) for more details.

At [www.PacktPub.com](http://www.PacktPub.com), you can also read a collection of free technical articles, sign up for a range of free newsletters and receive exclusive discounts and offers on Packt books and eBooks.



<http://PacktLib.PacktPub.com>

Do you need instant solutions to your IT questions? PacktLib is Packt's online digital book library. Here, you can access, read, and search across Packt's entire library of books.

## Why Subscribe?

- Fully searchable across every book published by Packt
- Copy and paste, print and bookmark content
- On demand and accessible via web browser

## Free Access for Packt account holders

If you have an account with Packt at [www.PacktPub.com](http://www.PacktPub.com), you can use this to access PacktLib today and view nine entirely free books. Simply use your login credentials for immediate access.

## Instant Updates on New Packt Books

Get notified! Find out when new books are published by following [@PacktEnterprise](https://twitter.com/PacktEnterprise) on Twitter, or the *Packt Enterprise* Facebook page.



# Table of Contents

<b>Preface</b>	<b>1</b>
<b>Chapter 1: Visual Studio 2012 Test Types</b>	<b>7</b>
<b>Software testing in Visual Studio 2012</b>	<b>8</b>
<b>Testing as part of software development life cycle</b>	<b>9</b>
<b>Types of testing</b>	<b>11</b>
Unit testing	12
Manual testing	14
Exploratory testing	15
Web performance tests	16
Coded UI test	17
Load testing	18
Ordered test	20
Generic test	21
<b>Test management in Visual Studio 2012</b>	<b>21</b>
Introduction to testing tools	22
Test Explorer	25
Code coverage results	28
<b>Microsoft Test Manager</b>	<b>28</b>
Connecting to Team Project	29
Test Plans, Suites, and test cases	30
Defining test cases	31
Lab Center	32
<b>Summary</b>	<b>33</b>
<b>Chapter 2: Test Plan, Test Suite, and Manual Testing</b>	<b>35</b>
<b>Test Plan</b>	<b>36</b>
<b>Test Suite and its types</b>	<b>41</b>
Static Test Suites	42
Query-based Test Suites	44
Requirement-based Test Suites	45

<b>Running manual tests</b>	<b>47</b>
Action recording	56
<b>Shared steps and action recording for shared steps</b>	<b>59</b>
Creating shared steps	59
Action recording for shared steps	62
<b>Adding parameters to manual tests</b>	<b>62</b>
<b>Summary</b>	<b>66</b>
<b>Chapter 3: Automated Tests</b>	<b>67</b>
<b>Coded UI tests from action recordings</b>	<b>68</b>
Files generated for coded UI test	73
CodedUITest1.cs	73
UIMap.Designer.cs	74
UIMap.cs	75
UiMap.uitest	76
Data-driven coded UI test	80
Adding controls and validation to coded UI test	82
<b>Summary</b>	<b>88</b>
<b>Chapter 4: Unit Testing</b>	<b>89</b>
<b>Creating unit tests</b>	<b>90</b>
<b>Assert statements</b>	<b>93</b>
Types of Assert statements	94
Assert	94
StringAsserts	107
CollectionAssert	111
AssertFailedException	119
UnitTestAssertionException	120
ExpectedExceptionAttribute	120
<b>Unit Tests and Generics</b>	<b>123</b>
<b>Data-driven unit testing</b>	<b>126</b>
<b>Unit Testing using Fakes</b>	<b>132</b>
Stubs	132
Shims	137
Difference between Stubs and Shims	137
<b>Code coverage unit test</b>	<b>138</b>
Blocks and lines	140
Excluding elements	141
<b>Summary</b>	<b>142</b>
<b>Chapter 5: Web Performance Test</b>	<b>143</b>
<b>Creating the web performance test</b>	<b>145</b>
Recording a test	146
Adding comments	152
Cleaning the recorded tests	153

---

Copying the requests	153
Adding loops	153
<b>Web performance test editor</b>	<b>158</b>
Web test properties	160
Web test request properties	161
Other request properties	164
Form POST parameters	164
QueryString parameters	165
Extraction rules	166
Validation rules	171
Transactions	174
Conditional rules	176
Toolbar properties	181
Add data source	181
Setting credentials	184
Add recording	185
Parameterize web server	186
Adding a web test plugin	189
<b>Debugging and running the web test</b>	<b>191</b>
Settings in the .testsettings file	192
General	192
Roles	194
Data and Diagnostics	195
Deployment	197
Setup and Cleanup Scripts	198
Hosts	199
Test Timeouts	199
Unit test	200
Web test	201
Running the test	203
Web Browser	204
Request	204
Response	205
Context	205
Details	206
<b>Summary</b>	<b>207</b>
<b>Chapter 6: Advanced Web Testing</b>	<b>209</b>
<b>    Dynamic parameters in web testing</b>	<b>210</b>
<b>    Coded web test</b>	<b>212</b>
Generating code from a recorded test	213
Transactions in coded tests	218
Custom code	219
Adding a comment	219
Running the coded web test	220
<b>    Debugging coded web test</b>	<b>222</b>

<b>Custom rules</b>	<b>224</b>
Extraction rules	224
Validation rules	228
<b>Summary</b>	<b>232</b>
<b>Chapter 7: Load Testing</b>	<b>233</b>
<b>Creating a Load Test</b>	<b>234</b>
Load Test Wizard	236
Specifying a scenario	239
Counter sets	248
Run settings	250
Editing Load Tests	262
Adding context parameters	267
Storing results in the central result store	268
Running the Load Test	270
Analyzing and exporting Test Results	272
Graphical view	272
Summary view	275
Tables view	277
Detail view	279
Exporting to Microsoft Excel	280
Using Test Controller and Test Agents	288
Test Controller and Test Agent configuration	289
<b>Summary</b>	<b>296</b>
<b>Chapter 8: Ordered and Generic Tests</b>	<b>297</b>
<b>Ordered tests</b>	<b>298</b>
Creating an ordered test	298
Executing an ordered test	300
<b>Generic tests</b>	<b>301</b>
Creating a generic test	302
The summary results file	304
<b>Summary</b>	<b>308</b>
<b>Chapter 9: Managing and Configuring Tests</b>	<b>309</b>
<b>Using Test settings</b>	<b>310</b>
The General option	311
The Roles option	312
Data and Diagnostics	313
The Deployment section	316
Setup and Cleanup Scripts	317
The Hosts option	318
The Test Timeouts option	319
The Unit Test option	320
Editing the Test Run configuration file	322

---

The Web Test option	324
Configuring unit tests using the .runsettings file	325
<b>Summary</b>	<b>326</b>
<b>Chapter 10: The Command Line</b>	<b>327</b>
<hr/>	
<b>VSTest.Console utility</b>	<b>327</b>
Running tests using VSTest.Console	328
The /Tests option	329
The /ListTests option	329
<b>MSTest utility</b>	<b>330</b>
Running a test from the command line	332
The /testcontainer option	332
The /testmetadata option	333
The /test option	334
The /unique option	335
The /noisolation option	336
The /testsettings option	336
The /resultsfile option	337
The /noresults option	337
The /nologo option	338
The /detail option	338
Publishing Test Results	339
The /publish option	339
The /publishbuild option	339
The /flavour option	340
The /platform option	340
The /publishresultsfile option	341
TCM command line utility	344
Importing tests to a Test Plan	345
Running tests in a Test Plan	349
<b>Summary</b>	<b>352</b>
<b>Chapter 11: Working with Test Results</b>	<b>353</b>
<hr/>	
<b>Test Runs and Test Results</b>	<b>354</b>
Test as part of the Team Foundation Server build	358
Building reports and Test Results	363
Creating a work item from the result	365
<b>Summary</b>	<b>367</b>
<b>Chapter 12: Exploratory Testing and Reporting</b>	<b>369</b>
<hr/>	
<b>Exploratory testing</b>	<b>371</b>
<b>Reports using Team Foundation Server</b>	<b>379</b>
Bug status report	379
Test case readiness report	379
Status on all iterations	380
Other out-of-the-box reports	380

<b>Creating a report definition using Visual Studio 2012</b>	<b>382</b>
<b>Summary</b>	<b>390</b>
<b>Chapter 13: Test and Lab Center</b>	<b>391</b>
<hr/>	
<b>Connecting to Team Project</b>	<b>392</b>
<b>Testing Center</b>	<b>394</b>
Testing Center – Plan tab	395
Testing Center – Test tab	399
Testing Center – Track tab	402
Testing Center – Organize tab	405
<b>Lab Center</b>	<b>408</b>
Environments	408
Deployed environments	410
<b>Summary</b>	<b>413</b>
<b>Index</b>	<b>415</b>
<hr/>	

# Preface

The Microsoft Visual Studio 2012 suite contains several features to support the needs of developers, testers, architects, and managers to simplify the development process. Visual Studio 2012 provides different editions of the product such as Professional, Premium, and Ultimate with different set of tools and features. Visual Studio 2012 is tightly integrated with Team Foundation Server, a central repository and configuration management system that provides version control, process guidance and templates, automated builds, automated tests, bug tracking, work item tracking, reporting, and support of the Lab Center and Test Center configurations. The Microsoft Test Manager 2012 is a standalone tool used to organize Test Plans, Manage test cases, and executing manual test cases.

*Software Testing using Visual Studio 2012* helps software developers to get familiarized with the Visual Studio tools and techniques to create automated unit tests, and to use automated user interface testing, code analysis and profiling to find out more about the performance and quality of the code. Testers benefit from learning more about the usage of Testing tools, test case management techniques, working with Test Results, and using Test Center and Lab center. This book also covers different types of testing such as web performance test, load test, executing the manual test cases, recording user actions, re-running tests using recording, test case execution, capturing defects, and linking defects with requirements. Testers also get a high level overview on using Lab Center for creating virtual environments for testing multiple users and multiple location scenarios.

Visual Studio 2012 provides user interface tools such as Test Explorer, Test Results, and Test Configuration to create, execute, and maintain the tests and Test Results in integration with Team Foundation Server. This book provides detailed information on all of the tools used for testing the application during the development and testing phases of the project life cycle.

## What this book covers

*Chapter 1, Visual Studio 2012 Test Types*, provides an overview of different types of testing which helps testing the software applications through different phases of software development. This chapter also introduces the tools and techniques in Visual Studio 2012 for different testing types, Microsoft Test Manager 2012, and its features.

*Chapter 2, Test Plan, Test Suite, and Manual Testing*, explains the steps involved in creating and managing the Test Plan, Test cases and Test Suite using Test Center in Test Manager. This chapter also explains how to create manual tests by recording the user actions and running the test with data inputs. Sharing the test recording across multiple tests is also covered in this chapter.

*Chapter 3, Automated Tests*, provides a step-by-step approach to creating Coded UI test from user action recordings. It also explains the steps to execute the coded UI test through data source and adding validation and custom rules to the test.

*Chapter 4, Unit Testing*, explains the detailed steps involved in creating unit test classes and methods for the code. Different type of assert methods and parameters for testing the code, passing set of data from a data source and testing the code also explained in detail. The mocking framework used for isolating the code and testing it with the help of Shims and Stubs is also explained in detail.

*Chapter 5, Web Performance Test*, explains the basic way of web testing by recording the user actions and creating a test out of it. Running the test using a data source, adding parameters to the web tests, adding validation and extraction rules, adding looping and branching mechanism to the recorded tests, and here configuring the settings required for the Test Runs are some of the features explained as part of this chapter.

*Chapter 6, Advanced Web Testing*, explains the way of generating code out of the recorded web tests explained in *Chapter 5, Web Performance Test* using the Generate Code option. This is very much useful for customizing the test through the code, adding additional logic to the test, adding custom validation and extraction rules.

*Chapter 7, Load Testing*, helps in simulating various numbers of users, network bandwidths, combination of different web browsers, and different configurations. In the case of web applications it is always necessary to test the stability and performance of the application under huge data load and concurrent users. This chapter explains the steps involved in simulating the real world scenario by using Controllers and Agents. The details of analyzing and exporting the load Test Results are also explained in this chapter.

*Chapter 8, Ordered and Generic Tests*, explains the way of testing the existing third party tool or service which can also be run using the command line. Visual Studio 2012 provides a feature called ordered test to group all or some of these tests and then execute the tests in the same order. The main advantage of creating the ordered test is to execute multiple tests in an order based on the dependencies. Generic tests are just like any other tests except that it is used for testing an existing third party tool or service.

*Chapter 9, Managing and Configuring Tests*, explains the details of the test settings file and the tools used for managing tests. The configuration includes deployment details, setup and cleaning scripts, collecting data diagnostics information, unit test and web test settings.

*Chapter 10, The Command Line*, explains the command line tools such as VSTest.Console, MSTest, and TCM used for running the test with different options, then collecting the output and publishing the results. Each of these commands is used for specific purposes including backwards compatibility.

*Chapter 11, Working with Test Results*, explains the process of running the tests and publishing the Test Results to the Team Project. Also covered in detail is to integrate the tests as part of Team Foundation Server builds, Build reports and Test Results, Creating work items from Test Results, and publishing the Test Results.

*Chapter 12, Exploratory Testing and Reporting*, explains the details of testing which happens without any test cases and scripts and by only exploring the application manually. This chapter also explains the details of accessing the Test Results and publishing Test Results and reporting the same in a specific format. Accessing different types of testing reports and creating new test reports are also explained in this chapter.

*Chapter 13, Test and Lab Center*, is useful for creating and organizing Test Plans and test cases. Test plans can be associated to the requirements using Test Center. The Lab Center helps in creating and configuring different virtual/physical environments for the Test Runs, Test Settings such as defining the roles and configuring the data and diagnostics information for the selected roles, configuring the Test Controllers required for the test, and configuring the test library to store the environment information.

## What you need for this book

This book requires a basic knowledge on any of the versions of Visual Studio and Team Foundation Server. The reader must be familiar with the Visual Studio IDE and have basic knowledge of C#. The following tools are required in order to use the code samples of the chapters in this book:

- Visual Studio 2012 Ultimate
- SQL Server Express (OR) SQL Server 2008 or higher version
- Team Foundation Server 2010/2012
- SQL Server Reporting services

## Who this book is for

If you are a software developer, a tester or an architect who wishes to master the amazing range of features offered by Visual Studio 2012 for testing your software applications – then this book is for you.

This book assumes that you have a basic knowledge of testing software applications and have good work experience of using Visual Studio IDE.

## Conventions

In this book, you will find a number of styles of text that distinguish between different kinds of information. Here are some examples of these styles, and an explanation of their meaning.

Code words in text are shown as follows: “All the methods and classes generated for the unit testing are inherited from the `Microsoft.VisualStudio.TestTools.UnitTesting` namespace.”

A block of code is set as follows:

```
[DataSource("Microsoft.VisualStudio.TestTools.DataSource.CSV",
"|DataDirectory|\\data.csv", "data#csv", DataAccessMethod.Sequential),
DeploymentItem("data.csv"), TestMethod]
```

**New terms** and **important words** are shown in bold. Words that you see on the screen, in menus or dialog boxes for example, appear in the text like this: “The **Test Runs** window displays all the tests based on the results availability at the location”.

 Warnings or important notes appear in a box like this. 

 Tips and tricks appear like this. 

## Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this book – what you liked or may have disliked. Reader feedback is important for us to develop titles that you really get the most out of.

To send us general feedback, simply send an e-mail to [feedback@packtpub.com](mailto:feedback@packtpub.com), and mention the book title via the subject of your message.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide on [www.packtpub.com/authors](http://www.packtpub.com/authors).

## Customer support

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

## Downloading the example code

You can download the example code files for all Packt books you have purchased from your account at <http://www.packtpub.com>. If you purchased this book elsewhere, you can visit <http://www.packtpub.com/support> and register to have the files e-mailed directly to you.

## Errata

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our books – maybe a mistake in the text or the code – we would be grateful if you would report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this book. If you find any errata, please report them by visiting <http://www.packtpub.com/submit-errata>, selecting your book, clicking on the **errata submission form** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded on our website, or added to any list of existing errata, under the Errata section of that title. Any existing errata can be viewed by selecting your title from <http://www.packtpub.com/support>.

## Piracy

Piracy of copyright material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works, in any form, on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy.

Please contact us at [copyright@packtpub.com](mailto:copyright@packtpub.com) with a link to the suspected pirated material.

We appreciate your help in protecting our authors, and our ability to bring you valuable content.

## Questions

You can contact us at [questions@packtpub.com](mailto:questions@packtpub.com) if you are having a problem with any aspect of the book, and we will do our best to address it.

# 1

## Visual Studio 2012 Test Types

Software testing is one of the most important phases of the **software development life cycle (SDLC)**. Delivery of the software product is based on following good SDLC practices of analysis, design, coding, testing, and by all means meeting the customer requirements. The quality of the product is measured by verifying and validating the product based on the defined functional and non-functional requirements for product. The testing tools and techniques play an important role in simulating the real-life scenarios and the user load required for verifying the stability and reliability of the product. For example, testing a web application with 1,000 concurrent users is a very time consuming and tedious task, if we do it manually considering the required resources. But the testing tools that are part of Visual Studio 2012 can simulate such scenarios and test it with limited resources and without manual intervention during testing. Visual Studio 2012 provides tools to conduct different types of testing, such as Unit testing, Load testing, Web testing, Ordered testing, Generic testing, and Exploratory testing.

This chapter covers the following topics and provides a high-level overview of the testing tools and techniques supported by Visual Studio 2012:

- Testing as part of the software development life cycle
- Types of testing
- Test management in Visual Studio 2012
- Testing tools in Visual Studio 2012

## Software testing in Visual Studio 2012

Before getting into the details of how to perform testing using Visual Studio 2012, let us familiarize different tools provided by Visual Studio 2012 and its usage. Visual Studio provides tools for testing as well as test management such as the Test List Editor and the Test View. The Test Projects and the actual test files are maintained in **Team Foundation Server (TFS)** for managing the version control of the source and history of changes.

The other aspect of this chapter is exploring the different file types generated in Visual Studio during testing. Most of these files are in the XML format, which are created automatically whenever a new test is created.

For readers new to Visual Studio, there is a brief overview on each window we are going to deal with throughout all or most of the chapters in this book. While we go through the windows and their purposes, we can check the **Integrated Development Environment (IDE)** and the tools integration with Visual Studio 2012.

Microsoft Visual Studio 2012 has different editions tailored to the needs. You need to have the respective edition as prerequisite to use any of the testing features explained in this book. The following table shows supported edition of Visual Studio 2012 for the testing features.

Testing features	Ultimate with MSDN	Premium with MSDN	Test Professional with MSDN	Professional with MSDN	Professional
Unit testing	Yes	Yes		Yes	Yes
Coded UI test	Yes	Yes			
Code coverage	Yes	Yes			
Manual testing	Yes	Yes	Yes		
Exploratory testing	Yes	Yes	Yes		
Test case management	Yes	Yes	Yes		

Testing features	Ultimate with MSDN	Premium with MSDN	Test Professional with MSDN	Professional with MSDN	Professional
Web performance testing	Yes				
Load testing	Yes				
Lab management	Yes	Yes	Yes		

**Microsoft Test Manager 2012 (MTM)** is a standalone product from Microsoft, which integrates with Team Foundation Server for test management. MTM is used in creating and managing multiple Test Plans, cloning Test Plans, creating Test Suites, creating manual test steps and test cases, and maintaining the same. MTM also provides various reports for Test Plan results. In 2012 version, MTM has the new feature of exploratory testing, maintaining records, and test steps during exploratory testing.

Lab environments can be created in MTM using the controller and agents. This is required when running load tests with multiple agents.

## Testing as part of software development life cycle

The main objective of testing is to find the early defects in the SDLC. If the defect is found early, then the cost will be lower than when the defect is found during the production or in the implementation stages. Moreover, testing is carried out to assure the quality and reliability of the software. In order to find the defect as soon as possible, the testing activities should start early, that is, in the Requirement phase of SDLC and continues till the end of the SDLC. The testing team should create the test cases based on the defined requirements.

The Coding phase of the SDLC includes various testing activities to validate and verify the functionality based on the design and the developer's code for the design. The developers themselves conduct the tests. In case of Test driven development, the test scripts and test scenarios are created first based on the requirement and the code is developed.

As soon as the developer completes the coding, the developer conducts the unit testing

- **Unit testing:** This is the first level of testing in the SDLC. The developer takes the smallest piece or unit of testable code and determines whether the code behaves exactly as expected. In object-oriented programming, the smallest unit is a method which belongs to a class. The method usually has one or few inputs and one output. Frameworks, drivers, Stubs and mock, or fake objects are used to assist in unit testing.

Once the coding is complete for the agreed requirements, all the units are integrated and the product is built as a single package. Then the other phases or forms of testing are executed.

- **Integration testing:** This type of testing is carried out between two or more modules or functions along with the intent of finding interface defects between them. This testing is completed as a part of unit or functional testing, and sometimes becomes its own standalone test phase. On a larger level, integration testing can involve putting together groups of modules and functions with the goal of completing and verifying that the system meets the system requirements. Defects found are logged and fixed later by the developers. There are different ways of integration testing such as top-down and bottom-up , which are as follows:
  - **Top-down approach:** This is the incremental testing technique which begins with the top level modules followed by low-level modules. The top-down approach helps in early detection of design errors which helps in saving development cost and time as the design errors can be fixed before implementation.
  - **Bottom-up approach:** This is exact opposite to the top-down approach. In this case the low level functionalities are tested and integrated first and then followed by the high level functionalities.
  - **Umbrella approach:** This approach uses both the top-down and bottom-up patterns. The inputs for functions are integrated in bottom-up approach and then the outputs for functions are integrated in the top-down approach.

- **System testing:** This type of testing is used for comparing or verifying the specifications against the developed system. The system test design is derived from the design documents and is used in this phase for planning and executing the tests. System testing is conducted after all the modules are integrated and completed with Integration testing. To avoid repeating the same process during multiple cycles of system testing, the tests are automated using automation testing tools. Once all the modules are integrated, several errors may arise because of dependencies and various other factors. The defects are usually maintained using a defect tracking tool and the development team prioritizes and fixes the defects. There are different types of testing followed under system testing, but they differ from organization to organization. Here are the common types of tests widely followed in the industry:
  - **Sanity testing:** Whenever there are some defect fixes to the existing product and because of that a new build is created, sanity test is conducted on that build instead of performing full testing on the software. Sanity test is conducted to make sure that the existing functionality of the product is not impacted or broken because of the defect fixes.
  - **Regression testing:** The main objective of this type is to determine if defect fixes or any other changes have been successful and have not introduced any new defects. This is also to verify if the existing functionalities are not affected.

## Types of testing

Visual Studio provides a range of testing types and tools for testing software applications. The following are some of those types:

- Unit test
- Manual test
- Exploratory test
- Web test
- Coded UI test
- Load test
- Ordered test
- Generic test

The unit testing tool is integrated along with Visual Studio and developers can use any of the Visual Studio supported language to write the unit testing. The manual test and exploratory test can be used during regression and is integrated with the Test Manager tool to track the test cases and defects when the test is conducted. Web Test and Coded UI Test in Visual Studio is used for system testing to record and playback the test steps. The load test tool is used during system testing cycle for testing performance and stability of the application with user load, and is integrated with Test Manager. The Generic test is again a part of the system testing to test the third-party components and the ordered test is to enable the testing order during Test Runs.

For all of the above testing types, Visual Studio provides tools to manage, order the listing, and execute tests. The next few sections provide details of these testing tools and the supporting tools for managing testing in Visual Studio 2012.

## **Unit testing**

Unit testing is one of the earliest phases of testing the application. In this phase the developers have to make sure that the unit of testable code delivers the expected output. It is extremely important to run unit tests to catch defects in the early stage of the software development cycle. The main goal of the unit testing is to isolate each piece of the code or individual functionality and test if individual method is returning the expected result for different sets of parameter values.

A unit test is a functional class method test by calling a method with the appropriate parameters, exercises it and compares the results with the expected outcome to ensure the correctness of the implemented code. Visual Studio 2012 has great support for unit testing through the integrated automated unit test framework, which enables developers to create and execute unit tests.

Visual Studio generates the test methods and the base code for the test methods. It is the responsibility of the developer to modify the generated test methods and customize the code for actual testing. The code file contains several attributes to identify the Test Class, Test Method, and Test Project. These attributes are assigned when the unit test code is created for the original source code. Here is the sample of the unit test code:

```
namespace UnitTestProject1
{
    [TestClass]
    public class UnitTest1
    {
        [TestMethod]
        public void TestAddNumbers()
        {
            string number1 = "10";
            string number2 = "8";
            double expectedTotal = 19;
            SampleClass sample = new SampleClass(number1, number2);
            double actualTotal = SampleClass.AddNumbers();
            Assert.AreEqual(expectedTotal, actualTotal, "The total is incorrect");
        }
    }
}
```

Once a unit test is created for a testable unit of code, the developers can use it with multiple combinations of input parameters to make sure the actual result is as per the expected result.

All the methods and classes generated for the unit testing are inherited from the `Microsoft.VisualStudio.TestTools.UnitTesting` namespace. This namespace is only used when the default Visual Studio integrated testing tool is used. This namespace contains many classes and attributes to provide enough information for the test engine to determine data source, test execution, execution order, deployment, and results.

Visual Studio also provides the flexibility to integrate unit testing tools such as Unit and XUnit for which the adapters need to be installed. After installing the tool, the respective namespaces can be used for generating calls and unit testing methods.

## Manual testing

Manual testing is the oldest and simplest type of testing, but yet very crucial for software testing. The tester would be writing the test cases based on the functional and non-functional requirements and then test the application based on each written test case. It helps us to validate whether the application meets various standards defined for effective and efficient accessibility and usage.

Manual testing can be an alternative in the following scenarios:

- The tests are more complex or too difficult to convert into automated tests.
- There is not enough time to automate the tests.
- Automated tests would be time consuming to create and run.
- There are not enough skilled resources to automate the tests.

The tested code hasn't stabilized sufficiently for cost effective automation.

We can create manual tests by using Visual Studio 2012 very easily. A very important step in manual testing is to document all the test steps required for the scenario with supporting information in a separate file. Once all the test cases are created, we should add the test cases to the Test Plan in order to run the test and gather the Test Result every time we run the test. The Microsoft Test Manager tool helps us in adding or editing the test cases to the Test Plan. The manual testing features supported by Visual Studio 2012 are as follows:

- Running the manual test multiple times with different data by changing parameters.
- Create multiple test cases using an existing test case and then customize or modify the test.
- Sharing test steps between multiple test cases.
- Remove the test cases from the test if no longer required.
- Adding or copying test steps from Microsoft Excel or Microsoft Word or from any other supported tool.
- Including multiple lines and rich text in manual test steps.

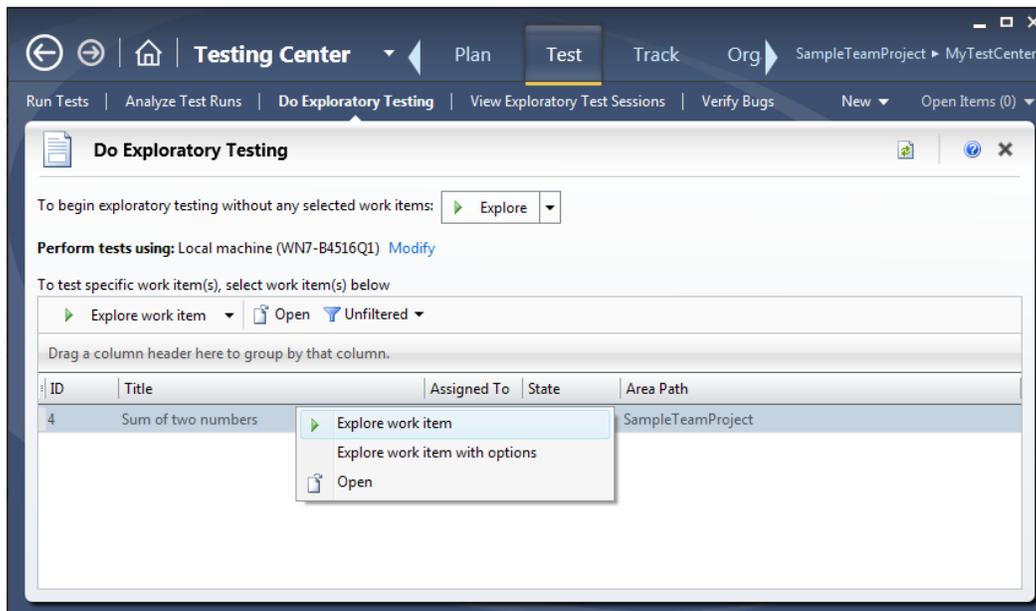
There are a lot of other manual testing features supported in Visual Studio 2012. We will see those features explained in *Chapter 2, Test Plan, Test Suite, and Manual Testing*.

## Exploratory testing

Exploratory testing is an open approach to testing without any process and test cases. The only known fact is the user story. The objective of this testing is to test the existing application or feature, and to find any improvements required, defects, broken links, and familiarize with the existing system. This type of testing has been followed for many years, but there was no tool to support the testing and capture the defects and steps. It was a tedious process to document the steps and capture supporting screenshots.

The **Microsoft Test Manager (MTM)** has the new feature to perform the exploratory testing and capture the screenshots, test steps, test case, comments, attachments, and defects automatically. The testing actions are stored as test cases so that it is easy while retesting.

To start exploratory testing, open the MTM and navigate to **Testing Center | Test | Do Exploratory Testing**. Now by selecting a work item requirement and then clicking on **Explore work item** will associate the recording of the test with the work item. Any test cases or defects created during Exploratory session will automatically get linked to the work item. The following screenshot shows a sample Exploratory testing session started for a work item:



During Exploratory testing, all actions performed on the screen are recorded except the actions performed in MTM and Office applications. To change this setting, configure the settings in the **Test Plan** properties.

A detailed walk-through the Exploratory testing is covered in *Chapter 12, Exploratory Testing and Reporting* which talks about Exploratory testing and reporting.

## Web performance tests

Web performance tests are used for testing the functionality and performance of the web page, web application, website, web services, and a combination of all of these. Web tests can be created by recording the HTTP requests and events during user interaction with the web application. The recording also captures the web page redirects, validations, view state information, authentication, and all the other activities. All these are possible through manually building the web tests using Web test. Visual Studio 2012 provides Web performance test features, which capture all HTTP requests and events while recording user interaction and generating the test.

There are different validation rules and extraction rules used in Web performance tests. The validation rules are used for validating the form field names, texts, and tags in the requested web page. We can validate the results or values against the expected result as per the business needs. These validation rules are also used for checking the processing time taken for the HTTP request.

Extraction rules in Web performance tests are used for collecting data from the web pages during requests and responses. The collection of these data will help us in testing the functionality and expected result from the response.

Providing sufficient data for the test methods is very important for the success of automated testing. Similarly for web tests we need to have a data source from which the data will be populated to the test methods and the web pages will be tested. The data source could be a database or a spread sheet or an XML data source or any other form of data source. There is a data binding mechanism in Web tests which takes care of fetching data from the source and provides the data to the test methods. For example, a reporting page in a web application definitely needs more data to test it successfully. This is also called the data-driven web test.

Web tests can be classified into Simple Web test and Coded Web test. Both of these are supported by Visual Studio.

- **Simple Web tests:** This includes generating and executing the test as per the recording with a valid flow of events. Once the test is started, there won't be any intervention and it won't be conditional.
- **Coded Web tests:** This is more complex, but provides a lot of flexibility. These types of tests are used for conditional execution based on certain values. Coded Web tests can be created manually or generated from a web test recording and languages such as C# or VB.NET can be chosen while generating the code. The generated code can be customized to better control the flow of test events. A coded Web test is a powerful and highly customizable test for the web requests.

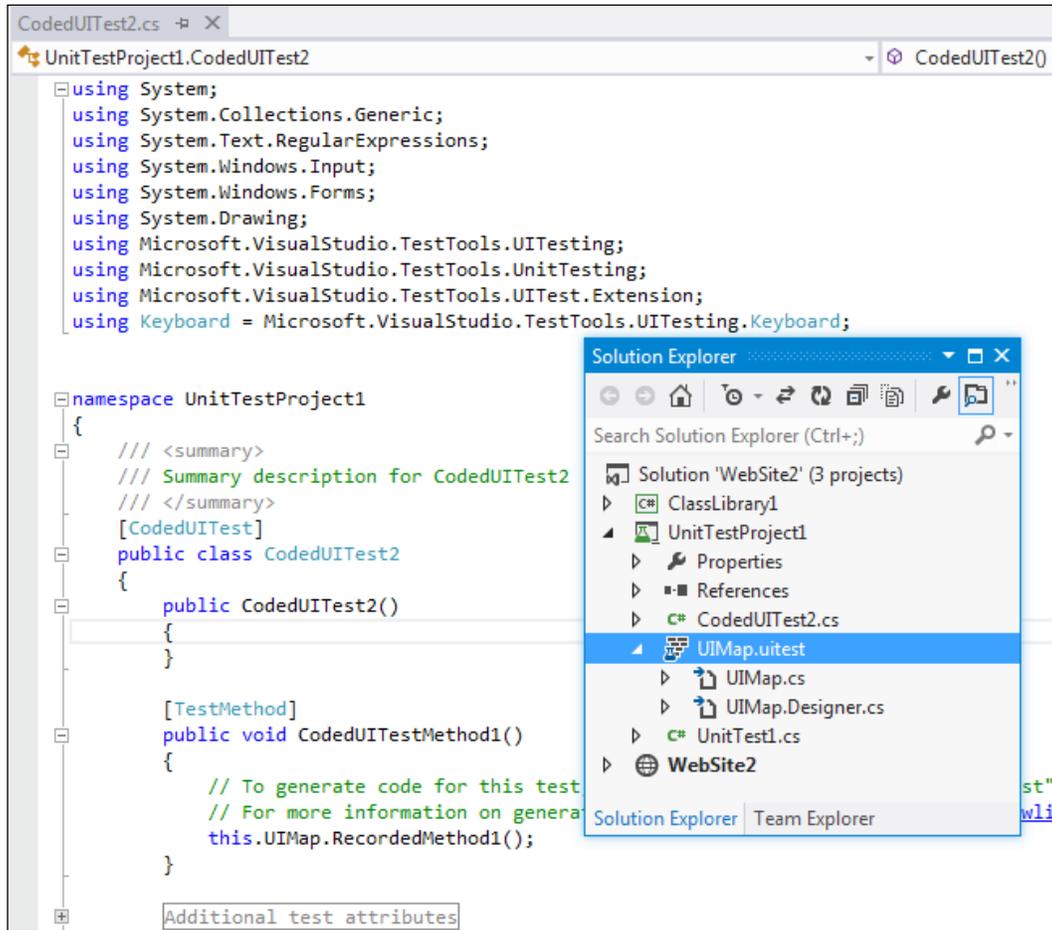
## Coded UI Test

**Coded UI Tests (CUIT)** are the automated way of testing the application user interface. In any UI intensive application, the functionality of the application is verified manually through UI and this happens after the development. Next time there is any change to any of the backend functionality, the application should be retested again. CUIT helps us in saving time spent testing through UI multiple times manually. CUIT Builder helps us in recording the UI test step actions and then generates code out of it. After the test is created, we can modify the code and customize the actions and data values captured during recording.

A Coded UI Test generates several supporting files as part of the testing. The `UIMap` object represents the controls, windows, and assertions. Using these objects and methods we can perform actions to automate the test. The coded UI Test supporting files are as follows:

- `CodedUITest.cs`: This file contains the test class, test methods, and assertions.
- `UIMap.uitest`: This is the XML model for `UIMap` class, which contains the windows, controls, properties, methods, and assertions.
- `UIMap.Designer.cs`: This contains the code for the `UIMap.uitest` XML file.
- `UIMap.cs`: All customization code for the UI Map would go into this file.

The following screenshot shows the Coded UI Test with the default files created for the test:



## Load testing

Load testing is a method of testing, which is used to identify the performance of the application under maximum workload. In case of a desktop or a standalone application, the user load is predictable, and thus easy to tune the performance, but in case of a multiuser application or a web application, it is required to determine the application behavior under normal and peak load conditions.

Visual Studio provides a load test feature, which helps in creating and executing load test with multiple scenarios. The following are the parameters set using the load test wizard:

- **Load Test Pattern:** This defines the number of users and the user load pattern to be followed during the test.
- **Test Mix Model:** This defines the model to be followed either by number of tests or by number of virtual users, or based on the user pace or by order.
- **Test Mix:** This includes the tests to be part of the load tests.
- **Browser Mix and Network Mix:** These define the possible browsers and the networks to follow while testing.
- **Counter Sets:** This defines the performance counters to collect from the load Test Agents and the system.
- **Run settings:** This defines the duration of the Test Run.

If the application is a public-facing website or one with a huge customer base, then it is better to perform load tests with real or expected scenarios. The Visual Studio load test makes use of the Web test recording or the unit test during load Test Run.

The load test is always driven by the collection of Web and Unit tests. A web test is used to simulate the scenario of concurrent users using the website and making multiple HTTP requests. The load can be configured to start with a minimum number of virtual users and then gradually increase the user count to check the performance at multiple stages of user load until it reaches the peak user load.

A unit test can be included as part of the load test in case of testing the performance of a service or individual method to find out the servicing capacity and threshold for client requests. One good example would be to test the data access service component that calls stored procedure from the backend database and returns the results to the client application.

The load test captures the results of individual tests within the Test Run. This helps us to identify the failed tests and debug and analyze them later. The results of all load tests can be saved in a repository to compare the set of results and then take necessary measures to improve performance.

Visual Studio has the Load test analyzer to provide the summary and details of Test Runs from the load Test Result.

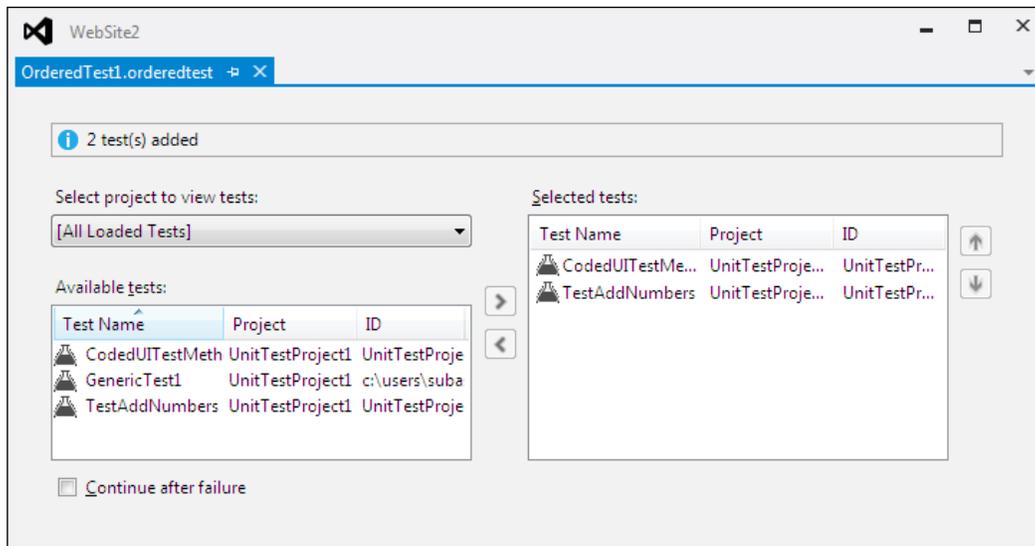
Load testing properties, working with tests, and analyzing the load Test Results are explained in detail later in this book in *Chapter 7, Load Testing*.

## Ordered test

Ordered test is just a container which holds the order in which a sequence of tests should be executed. All required tests should be ready and available to get added to the ordered test. Each test is independent and there is no dependency here. It is just the sequence of execution that is maintained in the ordered tests.

Test execution and results follow the sequence defined in the ordered test. The result of individual test is maintained in the repository. We can check the results anytime and analyze it.

Reordering the tests, adding new tests, and removing an existing test from the order are all possible through the Ordered Test Editor in Visual Studio.



An ordered test is the best way of controlling and running several tests in a defined order.

## **Generic test**

Generic test is useful in testing an existing executable file. It's the process of wrapping the executable file as a generic test and then executing it. This type of testing is very useful when testing a third party component without the source code. If the executable requires any additional files for testing, the same can be added as deployment files to the generic test. The test can be run using the Test Explorer or a command-line command.

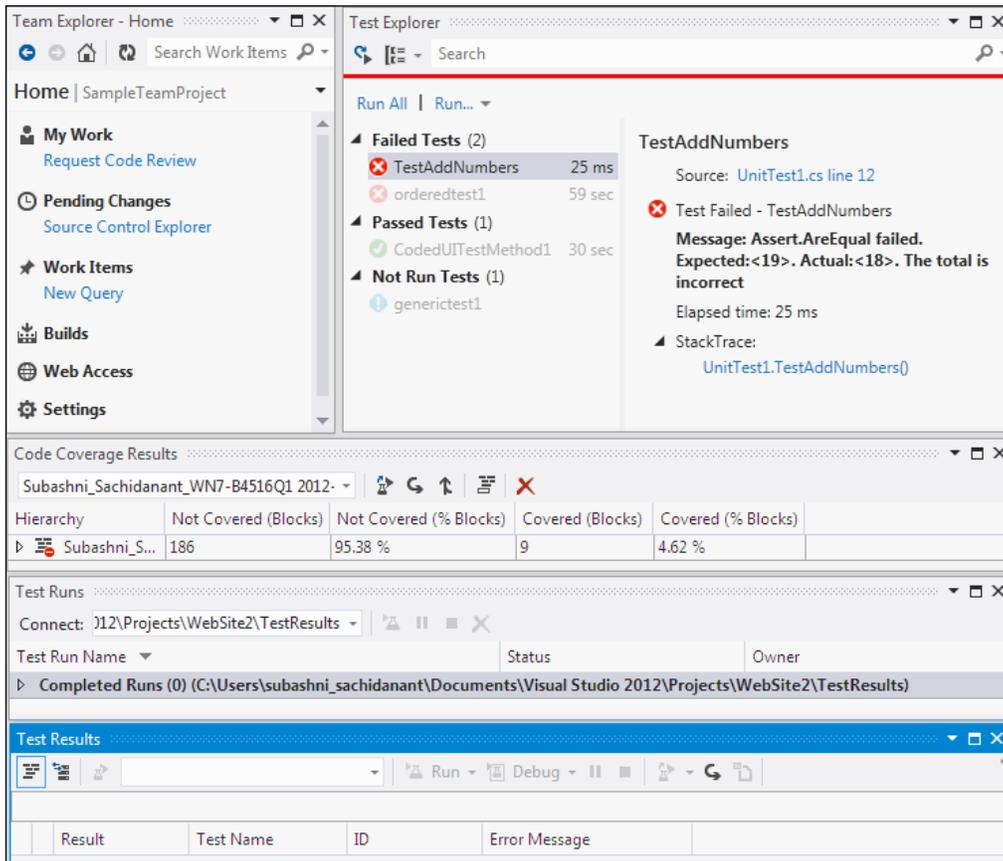
By using Visual Studio, we can collect the Test Results and gather code coverage data too. We can manage and run the generic tests in Visual Studio just like other tests. In fact, the Test Result output can be published to the Team Foundation Server to link it with the code built used for testing.

## **Test management in Visual Studio 2012**

Visual Studio has great testing features and management tools for testing. These features are greatly improved from previous versions of Visual Studio. The Test Impact View is the new test management tool added to the existing tools, such as Test View, Test List Editor, Test Results, Code Coverage Results, and Test Runs from the main IDE.

## Introduction to testing tools

Visual Studio provides tools to create, run, debug, and view results of your tests. The following screenshot is the overview of the tools and windows provided by Visual Studio for viewing the test and output details:

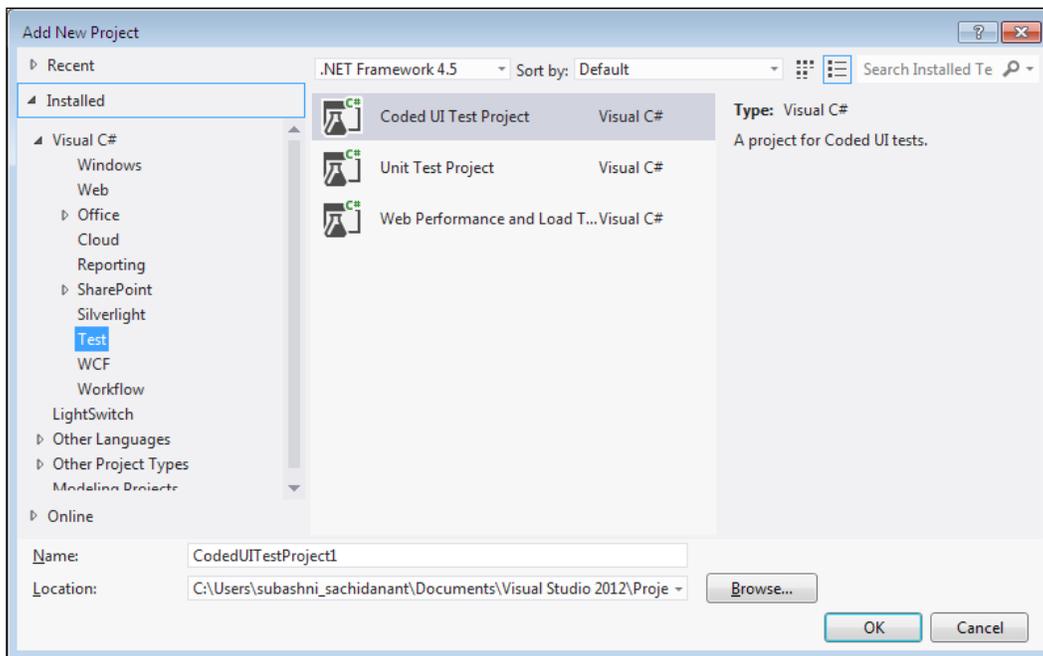


Let us create a new Test Project using Visual Studio 2012 and then test a sample project to get to know about the tools and features:

Open Visual Studio 2012 and create a new solution. Let's not get into the details of sample application, **AddNumbers**, but create the Test Project and look at the features of the tools and windows. The application referred throughout this chapter is a very simple application for adding two numbers and showing the result.

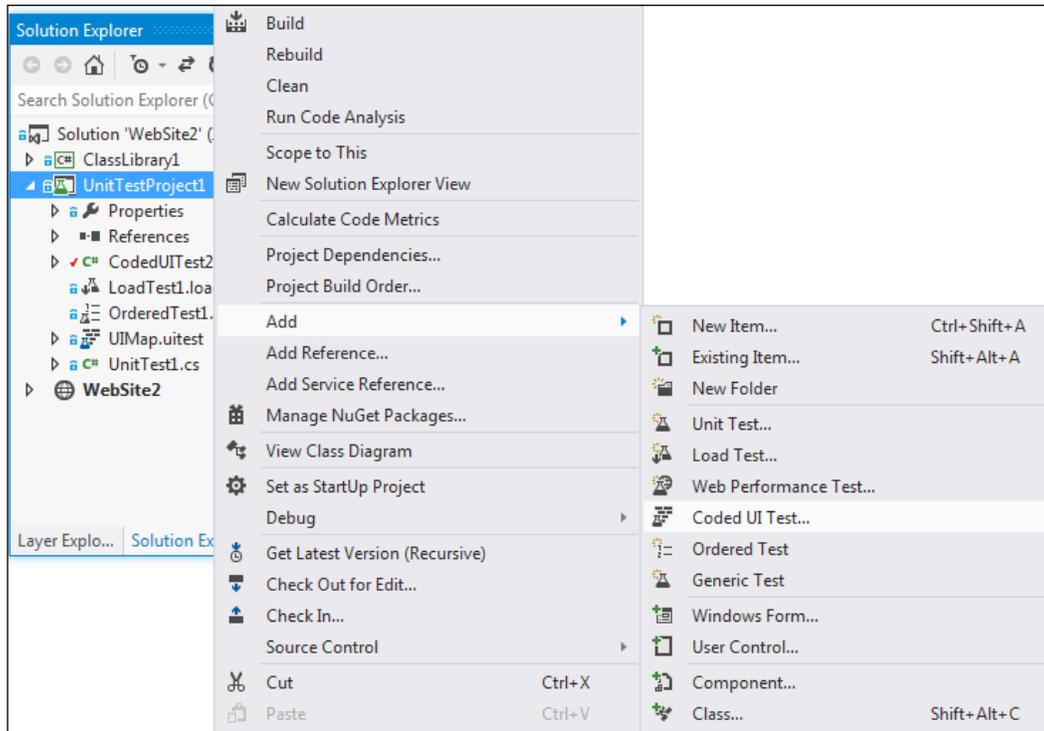
Now in a similar way to adding the projects and code files to the solution, create the Test Project and test files and add the Test Project to the solution.

Select the solution and add a project using the shortcut menu options **Add | New Project...** Then select the project type as **Test** from the list of project types under the language. Next select a template from the list. Visual Studio 2012 has three templates as follows:



For the sample testing application, select the second option, **Unit Test Project**. This option creates the project and also adds the unit test to the project.

The first option creates a Coded UI Test to capture the UI actions and controls, and automate the testing by generating and customizing the code. The last option is to record the user actions and re-run the recording to test with validations and verification rules, use the recording to test the performance and stability of the system under multiple user load.



The **Context** menu from the project has the option to choose new tests. The menu provides six different options for creating and adding the tests. The following is the file extension for each of the Visual Studio test types shown in the preceding screenshot:

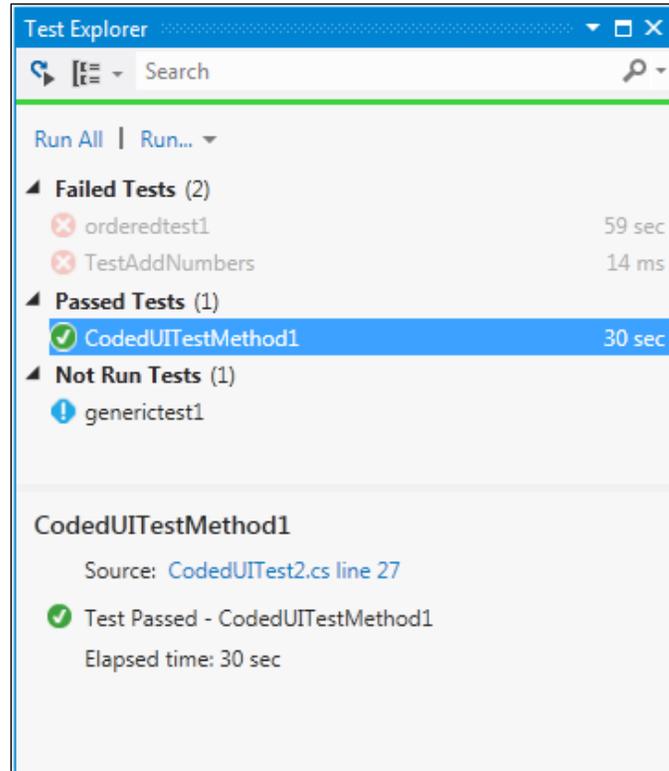
- .vb or .cs: This extension is for all types of **Unit Test** and **Coded UI Test**.
- .generictest: This extension is for the **Generic Test** type.
- .loadtest: This extension is for the **Load Test** type.
- .webtest: This extension is for the **Web Performance Test** type.

After selecting the test type, the test file gets created and added to the project with a default name and extension. Open the properties and change the name as required.

The next step is to use the **Test Explorer** window to view and run the tests that are created.

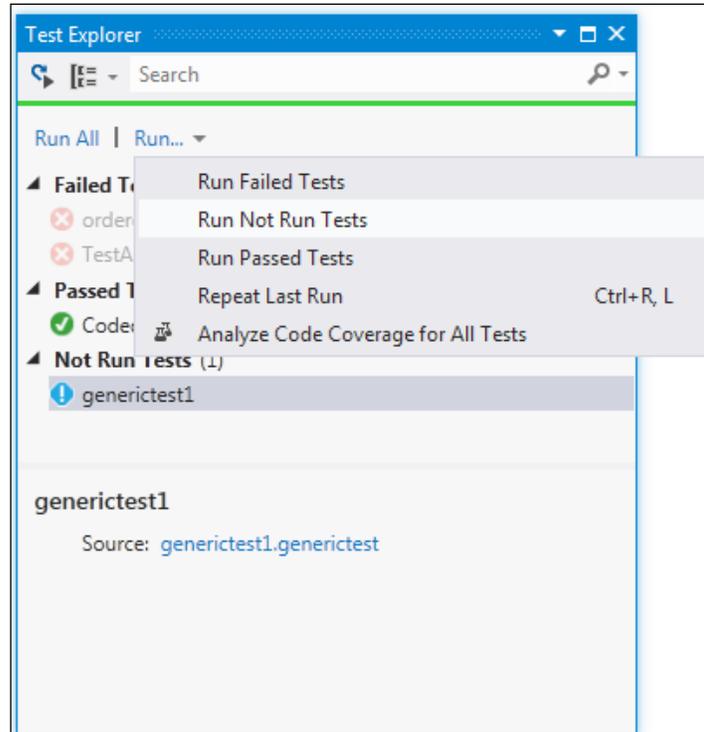
## Test Explorer

The **Test Explorer** window helps us to run tests from multiple projects in a solution. On building the Test Projects, the tests in each project appear in the **Test Explorer** window. To open **Test Explorer**, navigate to **Test | Windows | Test Explorer**. The tests are grouped into four different categories in **Test Explorer**, such as **Failed Tests**, **Passed Tests**, **Skipped Tests**, and **Not Run Tests** as shown in the following screenshot:



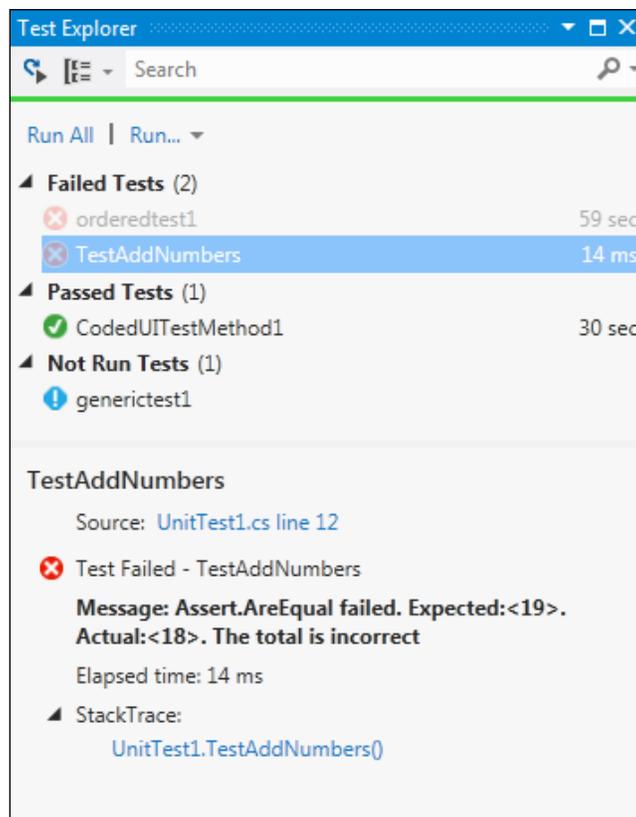
The **Test Explorer** window has the following options to run the tests:

- Choose **Run All...** to run all the tests in the solution
- Choose **Run...** and then a group to run all the tests under that group
- Select an individual test, open the **Context** menu, and then select **Run Selected Tests** to run only the selected tests



To view the details of the Test Run, select the test in the **Test Explorer** window. The **Details** pane displays the details as follows:

- **Source:** This is the source file name and the line number of the test method.
- **Status:** This is the test status whether it has passed or failed.
- **Message:** If the test is failed, the detailed message of the failure is also displayed.
- **Elapsed time:** This is the time that the method took to run.
- **StackTrace:** This is the stack trace information for the failed test.

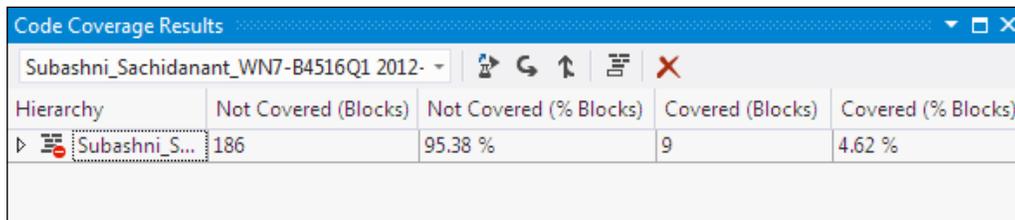


At any time if you double-click on the **Test** option or select **Test** and choose **Open Test**, Visual Studio opens the source code of the selected test. This is very helpful when starting to debug the code.

## Code coverage results

Visual Studio provides this code coverage feature to find out the percentage of code that is covered by the test execution. Through this window we can find out the number of lines covered by the test in each method.

Select the test from the **Test Explorer** window, and then right-click on the test and select **Analyze Code Coverage for Selected Tests**, or you can open the same by navigating to **Test | Windows | Code Coverage Results** from the **Menu** option. The following screenshot shows the code coverage results for the selected test. The result window provides information such as number of code blocks not covered by the test, percentage of code blocks not covered, covered code blocks, and the percentage of covered code blocks from the selected assembly:



Hierarchy	Not Covered (Blocks)	Not Covered (% Blocks)	Covered (Blocks)	Covered (% Blocks)
▶ Subashni_S...	186	95.38 %	9	4.62 %

## Microsoft Test Manager

This is the new standalone product introduced, but this is not a part of Visual Studio 2012 Premium. It is a part of Visual Studio Test Professional and Visual Studio Ultimate. This is the functional testing tool, which provides the ability to create and execute manual tests and collect the results. This tool works without Visual Studio but does require a connection to the Team Foundation Server and the Team Project.