

BEST PRACTICES

HOW WE TEST SOFTWARE AT MICROSOFT



Alan Page, Ken Johnston, Bj Rollison

How We Test Software at Microsoft®

*Alan Page
Ken Johnston
Bj Rollison*

PUBLISHED BY

Microsoft Press
A Division of Microsoft Corporation
One Microsoft Way
Redmond, Washington 98052-6399

Copyright © 2009 by Microsoft Corporation

All rights reserved. No part of the contents of this book may be reproduced or transmitted in any form or by any means without the written permission of the publisher.

Library of Congress Control Number: 2008940534

Printed and bound in the United States of America.

ISBN: 9780735624252

Second Printing: July 2014

Distributed in Canada by H.B. Fenn and Company Ltd.

A CIP catalogue record for this book is available from the British Library.

Microsoft Press books are available through booksellers and distributors worldwide. For further information about international editions, contact your local Microsoft Corporation office or contact Microsoft Press International directly at fax (425) 936-7329. Visit our Web site at www.microsoft.com/mspress. Send comments to mspinput@microsoft.com.

Microsoft, Microsoft Press, Access, Active Accessibility, Active Directory, ActiveX, Aero, Excel, Expression, Halo, Hotmail, Hyper-V, Internet Explorer, Microsoft Surface, MS, MSDN, MS-DOS, MSN, OneNote, Outlook, PowerPoint, SharePoint, SQL Server, Virtual Earth, Visio, Visual Basic, Visual Studio, Voodoo Vince, Win32, Windows, Windows Live, Windows Media, Windows Mobile, Windows NT, Windows Server, Windows Vista, Xbox, Xbox 360, and Zune are either registered trademarks or trademarks of the Microsoft group of companies. Other product and company names mentioned herein may be the trademarks of their respective owners.

The example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious. No association with any real company, organization, product, domain name, e-mail address, logo, person, place, or event is intended or should be inferred.

This book expresses the author's views and opinions. The information contained in this book is provided without any express, statutory, or implied warranties. Neither the authors, Microsoft Corporation, nor its resellers, or distributors will be held liable for any damages caused or alleged to be caused either directly or indirectly by this book.

Acquisitions Editor: Ben Ryan

Project Editor: Lynn Finnel

Editorial Production: Waypoint Press

Cover Illustration: John Hersey

Body Part No. X14-71546

To my wife, Christina, and our children, Cole and Winona, who sacrificed their time with me so I could write this book; and for my parents, Don and Arlene, for their constant support, and for giving me a sanctuary to write.

—Alan Page

To my children, David and Grace, for allowing their dad the time to write; and to my wife, Karen, who while I was working on a presentation for a testing conference first suggested, "Why don't you just call it 'How we test at Microsoft.'" Without those words (and Alan leading the way), we would not have started or finished this project.

—Ken Johnston

To my mother and father for their unending love, sage wisdom, and especially their patience. I also want to thank my 6-year-old daughter Elizabeth whose incessant curiosity to learn new things and persistent determination to conquer diverse challenges has taught me that the only problems we cannot overcome are those for which we have not yet found a solution.

—Bj Rollison

This book is dedicated to the test engineers at Microsoft who devote themselves to the most challenging endeavor in the software process, and who continue to mature the discipline by breaking through traditional barriers and roles in order to help ship leading-edge, high-quality software to our customers. For us, it is truly a privilege to mentor and work alongside so many professional testers at Microsoft, because through our interactions with them we also continue to learn more about software testing.

Contents at a Glance

Part I	About Microsoft	
1	Software Engineering at Microsoft	3
2	Software Test Engineers at Microsoft	21
3	Engineering Life Cycles	41
Part II	About Testing	
4	A Practical Approach to Test Case Design	61
5	Functional Testing Techniques	73
6	Structural Testing Techniques	115
7	Analyzing Risk with Code Complexity	145
8	Model-Based Testing	159
Part III	Test Tools and Systems	
9	Managing Bugs and Test Cases	187
10	Test Automation	219
11	Non-Functional Testing	249
12	Other Tools	273
13	Customer Feedback Systems	297
14	Testing Software Plus Services	317
Part IV	About the Future	
15	Solving Tomorrow's Problems Today	365
16	Building the Future	389

Table of Contents

Acknowledgments	xv
Introduction	xvii

Part I **About Microsoft**

1 Software Engineering at Microsoft	3
The Microsoft Vision, Values and Why We “Love This Company!”	3
Microsoft Is a Big Software Engineering Company	7
Developing Big and Efficient Businesses	8
The Shared Team Model	9
Working Small in a Big Company	11
Employing Many Types of Engineers	14
The Engineering Disciplines	15
Being a Global Software Development Company	17
Summary	20
2 Software Test Engineers at Microsoft	21
What’s in a Name?	23
Testers at Microsoft Have Not Always Been SDETs	24
I Need More Testers and I Need Them Now!	27
Campus Recruiting	29
Industry Recruiting	31
Learning How to Be a Microsoft SDET	32
The Engineering Career at Microsoft	33

 **What do you think of this book? We want to hear from you!**

Microsoft is interested in hearing your feedback so we can continually improve our books and learning resources for you. To participate in a brief online survey, please visit:

www.microsoft.com/learning/booksurvey

- Career Paths in the Test Discipline 34
 - The Test Architect 34
 - The IC Tester 35
 - Becoming a Manager is Not a Promotion..... 36
 - Test Managers 38
 - Summary 39
- 3 Engineering Life Cycles 41**
 - Software Engineering at Microsoft 41
 - Traditional Software Engineering Models 42
 - Milestones 45
 - Agile at Microsoft 48
 - Putting It All Together..... 50
 - Process Improvement 51
 - Formal Process Improvement Systems at Microsoft 52
 - Shipping Software from the War Room..... 54
 - Mandatory Practices 56
 - Summary: Completing the Meal 57

Part II About Testing

- 4 A Practical Approach to Test Case Design 61**
 - Practicing Good Software Design and Test Design 61
 - Using Test Patterns..... 62
 - Estimating Test Time 64
 - Starting with Testing 65
 - Ask Questions..... 65
 - Have a Test Strategy 66
 - Thinking About Testability 67
 - Test Design Specifications 68
 - Testing the Good and the Bad..... 69
 - Other Factors to Consider in Test Case Design..... 70
 - Black Box, White Box, and Gray Box..... 71
 - Exploratory Testing at Microsoft..... 71
 - Summary 72

5	Functional Testing Techniques	73
	The Need for Functional Testing	74
	Equivalence Class Partitioning	78
	Decomposing Variable Data	80
	Equivalence Class Partitioning in Action	83
	Analyzing Parameter Subsets	84
	The ECP Tests	86
	Summary of Equivalence Class Partitioning	89
	Boundary Value Analysis	90
	Defining Boundary Tests	92
	A New Formula for Boundary Value Analysis	93
	Hidden Boundaries	97
	Summary of Boundary Value Analysis	100
	Combinatorial Analysis	100
	Combinatorial Testing Approaches	101
	Combinatorial Analysis in Practice	104
	Effectiveness of Combinatorial Analysis	111
	Summary of Combinatorial Analysis	113
	Summary	113
6	Structural Testing Techniques	115
	Block Testing	118
	Summary of Block Testing	126
	Decision Testing	126
	Summary of Decision Testing	128
	Condition Testing	129
	Summary of Condition Testing	132
	Basis Path Testing	132
	Summary of Basis Path Testing	142
	Summary	142
7	Analyzing Risk with Code Complexity	145
	Risky Business	145
	A Complex Problem	146
	Counting Lines of Code	148
	Measuring Cyclomatic Complexity	149
	Halstead Metrics	152
	Object-Oriented Metrics	153

High Cyclomatic Complexity Doesn't Necessarily Mean "Buggy" 155
What to Do with Complexity Metrics 157
Summary 158

8 Model-Based Testing 159

Modeling Basics 160
Testing with Models. 161
 Designing a Model 161
 Modeling Software 162
 Building a Finite State Model. 166
 Automating Models. 166
Modeling Without Testing 172
 Bayesian Graphical Modeling 172
 Petri Nets. 173
Model-Based Testing Tools at Microsoft 174
 Spec Explorer 174
 A Language and Engine 179
 Modeling Tips. 182
Summary 183
Recommended Reading and Tools 183

Part III Test Tools and Systems

9 Managing Bugs and Test Cases. 187

The Bug Workflow 188
Bug Tracking 188
 A Bug's Life 189
 Attributes of a Bug Tracking System. 190
 Why Write a Bug Report? 191
 Anatomy of a Bug Report. 192
 Bug Triage. 196
 Common Mistakes in Bug Reports 198
 Using the Data 201
 How Not to Use the Data: Bugs as Performance Metrics 204
 Bug Bars. 205
Test Case Management. 209
 What Is a Test Case? 209
 The Value of a Test Case. 211
 Anatomy of a Test Case. 212
 Test Case Mistakes. 213

Managing Test Cases	215
Cases and Points: Counting Test Cases	215
Tracking and Interpreting the Test Results	217
Summary	218
10 Test Automation	219
The Value of Automation	219
To Automate or Not to Automate, That Is the Question	220
User Interface Automation	223
What's in a Test?	228
SEARCH at Microsoft	232
Setup	232
Execution	234
Analysis	240
Reporting	243
Cleanup	243
Help	244
Run, Automation, Run!	245
Putting It All Together	245
Large-Scale Test Automation	246
Common Automation Mistakes	247
Summary	248
11 Non-Functional Testing	249
Beyond Functionality	249
Testing the "ilities"	250
Performance Testing	252
How Do You Measure Performance?	253
Stress Testing	255
Distributed Stress Testing	257
Distributed Stress Architecture	258
Attributes of Multiclient Stress Tests	260
Compatibility Testing	261
Application Libraries	261
Application Verifier	262
Eating Our Dogfood	264
Accessibility Testing	265
Accessibility Personas	266
Testing for Accessibility	267
Testing Tools for Microsoft Active Accessibility	268

Usability Testing	269
Security Testing	270
Threat Modeling	271
Fuzz Testing	271
Summary	272
12 Other Tools	273
Code Churn	273
Keeping It Under Control	275
Tracking Changes	275
What Changed?	276
Why Did It Change?	278
A Home for Source Control	279
Build It	281
The Daily Build	281
Static Analysis	288
Native Code Analysis	288
Managed Code Analysis	290
Just Another Tool	291
Test Code Analysis	292
Test Code Is Product Code	293
Even More Tools	294
Tools for Unique Problems	294
Tools for Everyone	294
Summary	295
13 Customer Feedback Systems	297
Testing and Quality	297
Testing Provides Information	297
Quality Perception	298
Customers to the Rescue	299
Games, Too!	303
Windows Error Reporting	304
The Way We WER	305
Filling the Buckets	307
Emptying the Buckets	307
Test and WER	309

Smile and Microsoft Smiles with You	309
Send a Smile Impact	311
Connecting with Customers	312
Summary	315

14 Testing Software Plus Services 317

Two Parts: About Services and Test Techniques	318
Part 1: About Services	318
The Microsoft Services Strategy	318
Shifting to Internet Services as the Focus	319
Growing from Large Scale to Mega Scale	320
Power Is the Bottleneck to Growth	323
Services vs. Packaged Product	325
Moving from Stand-Alone to Layered Services	327
Part 2 Testing Software Plus Services	329
Waves of Innovation	329
Designing the Right S+S and Services Test Approach	330
Testing Techniques for S+S	337
Several Other Critical Thoughts on S+S	355
Continuous Quality Improvement Program	355
Common Bugs I've Seen Missed	359
Summary	361

Part IV About the Future

15 Solving Tomorrow's Problems Today 365

Automatic Failure Analysis	365
Overcoming Analysis Paralysis	365
The Match Game	367
Good Logging Practices	368
Anatomy of a Log File	370
Integrating AFA	371
Machine Virtualization	372
Virtualization Benefits	372
Virtual Machine Test Scenarios	374
When a Failure Occurs During Testing	377
Test Scenarios Not Recommended	379

Code Reviews and Inspections	379
Types of Code Reviews	380
Checklists	381
Other Considerations	381
Two Faces of Review	384
Tools, Tools, Everywhere	384
Reduce, Reuse, Recycle	385
What's the Problem?	385
Open Development	386
Summary	387
16 Building the Future	389
The Need for Forward Thinking	389
Thinking Forward by Moving Backward	389
Striving for a Quality Culture	390
Testing and Quality Assurance	391
Who Owns Quality?	392
The Cost of Quality	393
A New Role for Test	394
Test Leadership	394
The Microsoft Test Leadership Team	394
Test Leadership Team Chair	395
Test Leadership in Action	396
The Test Architect Group	397
Test Excellence	400
Sharing	400
Helping	401
Communicating	402
Keeping an Eye on the Future	404
Microsoft Director of Test Excellence	404
The Leadership Triad	404
Innovating for the Future	405
Index	407

What do you think of this book? We want to hear from you!

Microsoft is interested in hearing your feedback so we can continually improve our books and learning resources for you. To participate in a brief online survey, please visit:

www.microsoft.com/learning/booksurvey

Acknowledgments

This book never would have happened without the help of every single tester at Microsoft. Many helped directly by reviewing chapters or writing about their experiences in testing. Others helped by creating the legacy of software testing at Microsoft, or by continuing to innovate in the way we test software.

Including the names of all 9,000 testers at Microsoft would be impractical (especially when many former employees, people from other disciplines, and even external reviewers contributed to the completion of this book). On the other hand, we do want to call out the names of several people who have contributed substantially to the creation of this book.

This book grew out of the opinions, suggestions, and feedback of many current and former Microsoft employees. Some of the most prominent contributors include Michael Corning, Ed Triou, Amol Kher, Scott Wadsworth, Geoff Staneff, Dan Trivison, Brian Rogers, John Lambert, Sanjeev Verma, Shawn McFarland, Grant George, Tara Roth, Karen Carter-Schwendler, Jean Hartman, James Whittaker, Irada Sadykhova, Alex Kim, Darrin Hatakeda, Marty Riley, Venkat Narayanan, Karen Johnston, Jim Pierson, Ibrahim El Far, Carl Tostevin, Nachi Nagappan, Keith Stobie, Mark Davis, Mike Blaylock, Wayne Roseberry, Carole Cancler, Andy Tischaefer, Lori Ada-Kilty, Matt Heusser, Jeff Raikes, Microsoft Research (especially Amy Stevenson), the Microsoft Test Excellence Team, the Microsoft Test Leadership Team, and the Microsoft Test Architect Group.

We'd also like to thank Lynn Finnel, the Project Editor for this book, who continued to give us encouragement and support throughout the creation of this book.

Introduction

I still remember the morning, sometime late in the fall of 2007, when my manager at the time, Ken Johnston, uttered these five words, “You should write a book.”

He had just come back from delivering a talk at an industry test conference (not coincidentally titled, “How We Test Software at Microsoft,”) and was excited by the audience reception. Ken loves to give presentations, but he somehow thought I should be the one to write the book.

I humored him and said, “Sure, why not.” I went on to say that the book could cover a lot of the things that we teach in our software testing courses, as well as a smattering of other popular test approaches used at Microsoft. It could be interesting, but there are a ton of books on testing—I know, I’ve probably read a few dozen of them—and some of them are really good. What value to the testing community could yet another book provide?

I was about to talk the nonsense out of Ken when I realized something critical: At Microsoft, we have some of the best software test training in the world. The material and structure of the courses are fantastic, but that’s not what makes it so great. The way our instructors tie in anecdotes, success stories, and cool little bits of trivia throughout our courses is what makes them impactful and memorable. I thought that if we could include some stories and bits of information on how Microsoft has used some of these approaches, the book might be interesting. I began to think beyond what we teach, of more test ideas and stories that would be fun to share with testers everywhere. I realized that some of my favorite programming books were filled with stories embedded with all of the “techie” stuff.

The next thing I knew, I was writing a proposal. An outline began to come together, and the form of the book began to take shape, with four main themes emerging. It made sense to set some context by talking about Microsoft’s general approaches to people and engineering. The next two sections would focus on how we do testing inside Microsoft, and the tools we use; and the final section would look at the future of testing inside Microsoft. I sent the proposal to Microsoft Press, and although I remained excited about the potential for the book, part of me secretly hoped that Microsoft Press would tell me the idea was silly, and that I should go away. Alas, that didn’t happen, and shortly thereafter, I found myself staring at a computer screen wondering what the first sentences would look like.

From the very beginning, I knew that I wanted Ken to write the first two chapters. Ken has been a manager at Microsoft for years, and the people stuff was right up his alley. About the time I submitted the proposal, Ken left our group to manage the Office Online group. Soon after, it became apparent that Ken should also write the chapter on how we test Software plus Services. He’s since become a leader at the company in defining how we test Web services, and it would have been silly not to have him write Chapter 14, “Testing Software

Plus Services". Later on, I approached BJ Rollison, one of Microsoft's most prominent testers, to write the chapters about functional and structural test techniques. BJ Rollison designed our core software testing course, and he knows more about these areas of testing than anyone I know. He's also one of the only people I know who has read more books on testing than I have. Ken, Bj and I make quite a trio of authors. We all approach the task and produce our material quite differently, but in the end, we feel like we have a mix of both material and writing styles that reflects the diversity of the Microsoft testing population. We often joke that Bj is the professor, Ken tries to be the historian and storyteller, and I just absorb information and state the facts. Although we all took the lead on several chapters, we each edited and contributed to the others' work, so there is definitely a melding of styles throughout the book.

I cannot begin to describe how every little setback in life becomes gigantic when the task of "writing a book" is always on your plate. Since starting this book, I took over Ken's old job as Director of Test Excellence at Microsoft. Why in the world I decided to take on a job with entirely new challenges in the middle of writing a book I'll never know. In hindsight, however, taking on this role forced me to gain some insight into test leadership at Microsoft that benefitted this book tremendously.

My biggest fear in writing this book was how much I knew I'd have to leave out. There are over 9,000 testers at Microsoft. The test approaches discussed in this book cover what most testers at Microsoft do, but there are tons of fantastically cool things that Microsoft testers do that couldn't be covered in this book. On top of that, there are variations on just about every topic covered in this book. We tried to capture as many different ideas as we could, while telling stories about what parts of testing we think are most important. I also have to admit that I'm slightly nervous about the title of this book. "How We Test Software at Microsoft" could imply that everything in this book is done by every single tester at Microsoft, and that's simply not true. With such a large population of testers and such a massive product portfolio, there's just no way to write about testing in a way that exactly represents every single tester at Microsoft. So, we compromised. This book simply covers the most popular testing practices, tools, and techniques used by Microsoft testers. Not every team does everything we write about, but most do. Everything we chose to write about in this book has been successful in testing Microsoft products, so the topics in this book are a collection of some of the things we know work.

In the end, I think we succeeded, but as testers, we know it could be better. Sadly, it's time to ship, but we do have a support plan in place! If you are interested in discussing anything from this book with the authors, you can visit our web site, www.hwtsam.com. We would all love to hear what you think.

—Alan Page

Who This Book Is For

This book is for anyone who is interested in the role of test at Microsoft or for those who want to know more about how Microsoft approaches testing. This book isn't a replacement for any of the numerous other great texts on software testing. Instead, it describes how Microsoft applies a number of testing techniques and methods of evaluation to improve our software.

Microsoft testers themselves will likely find the book to be interesting as it includes techniques and approaches used across the company. Even nontesters may find it interesting to know about the role of test at Microsoft

What This Book Is About

This book starts by familiarizing the reader with Microsoft products, Microsoft engineers, Microsoft testers, the role of test, and general approaches to engineering software. The second part of the book discusses many of the test approaches and tools commonly used at Microsoft. The third part of the book discusses some of the tools and systems we use in our work. The final section of the book discusses future directions in testing and quality at Microsoft and how we intend to create that future.

Part I, "About Microsoft"

Chapter 1, "Software Engineering at Microsoft,"

Chapter 2, "Software Test Engineers at Microsoft"

Chapter 3, "Engineering Life Cycles"

Part II, "About Testing"

Chapter 4, "A Practical Approach to Test Case Design"

Chapter 5, "Functional Testing Techniques"

Chapter 6, "Structural Testing Techniques"

Chapter 7, "Analyzing Risk with Code Complexity"

Chapter 8, "Model-Based Testing"

Part III, "Test Tools and Systems"

Chapter 9, "Managing Bugs and Test Cases"

Chapter 10, "Test Automation"

Chapter 11, "Non-Functional Testing"

Chapter 12, "Other Tools"

Chapter 13, "Customer Feedback Systems"

Chapter 14, "Testing Software Plus Services"

Part IV, "About the Future"

Chapter 15, "Solving Tomorrow's Problems Today"

Chapter 16, "Building the Future"

Find Additional Content Online

As new or updated material becomes available that complements this book, it will be posted online on the Microsoft Press Online Developer Tools Web site. The type of material you might find includes updates to book content, articles, links to companion content, errata, sample chapters, and more. This Web site is available at www.microsoft.com/learning/books/online/developer, and is updated periodically.

More stories and tidbits about testing at Microsoft will be posted on www.hwtsam.com.

Support for This Book

If you have comments, questions, or ideas regarding the book, or questions that are not answered by visiting the sites above, please send them to Microsoft Press via e-mail to

mspinput@microsoft.com.

Or via postal mail to

Microsoft Press

Attn: *How We Test Software at Microsoft* Editor

One Microsoft Way

Redmond, WA 98052-6399.

Please note that Microsoft software product support is not offered through the above addresses.



Chapter 3

Engineering Life Cycles

Alan Page

I love to cook. Something about the entire process of creating a meal, coordinating multiple dishes, and ensuring that they all are complete at the exact same time is fun for me. My approach, learned from my highly cooking-talented mother, includes making up a lot of it as I go along. In short, I like to “wing it.” I’ve cooked enough that I’m comfortable browsing through the cupboard to see which ingredients seem appropriate. I use recipes as a guideline—as something to give me the general idea of what kinds of ingredients to use, how long to cook things, or to give me new inspiration. There is a ton of flexibility in my approach, but there is also some amount of risk. I might make a poor choice in substitution (for example, I recommend that you never replace cow’s milk with soymilk when making strata).

My approach to cooking, like testing, depends on the situation. For example, if guests are coming for dinner, I might measure a bit more than normal or substitute less than I do when cooking just for my family. I want to reduce the risk of a “defect” in the taste of my risotto, so I put a little more formality into the way I make it. I can only imagine the chef who is in charge of preparing a banquet for a hundred people. When cooking for such a large number of people, measurements and proportions become much more important. In addition, with such a wide variety of taste buds to please, the chef’s challenge is to come up with a combination of flavors that is palatable to all of the guests. Finally, of course, the entire meal needs to be prepared and all elements of the meal need to be freshly hot and on the table exactly on time. In this case, the “ship date” is unchangeable!

Making software has many similarities with cooking. There are benefits to following a strict plan and other benefits that can come from a more flexible approach, and additional challenges can occur when creating anything for a massive number of users. This chapter describes a variety of methods used to create software at Microsoft.

Software Engineering at Microsoft

There is no “one model” that every product team at Microsoft uses to create software. Each team determines, given the size and scope of the product, market conditions, team size, and prior experiences, the best model for achieving their goals. A new product might be driven by time to market so as to get in the game before there is a category leader. An established product might need to be very innovative to unseat a leading competitor or to stay ahead of the pack. Each situation requires a different approach to scoping, engineering, and shipping the product. Even with the need for variation, many practices and approaches have

become generally adopted, while allowing for significant experimentation and innovation in engineering processes.

For testers, understanding the differences between common engineering models, the model used by their team, and what part of the model their team is working in helps both in planning (knowing what will be happening) and in execution (knowing the goals of the current phase of the model). Understanding the process and their role in the process is essential for success.

Traditional Software Engineering Models

Many models are used to develop software. Some development models have been around for decades, whereas others seem to pop up nearly every month. Some models are extremely formal and structured, whereas others are highly flexible. Of course, there is no single model that will work for every software development team, but following some sort of proven model will usually help an engineering team create a better product. Understanding which parts of development and testing are done during which stages of the product cycle enables teams to anticipate some types of problems and to understand sooner when design or quality issues might affect their ability to release on time.

Waterfall Model

One of the most commonly known (and commonly abused) models for creating software is the waterfall model. Waterfall is an approach to software development where the end of each phase coincides with the beginning of the next phase, as shown in Figure 3-1. The work follows steps through a specified order. The implementation of the work “flows” from one phase to another (like a waterfall flows down a hill).

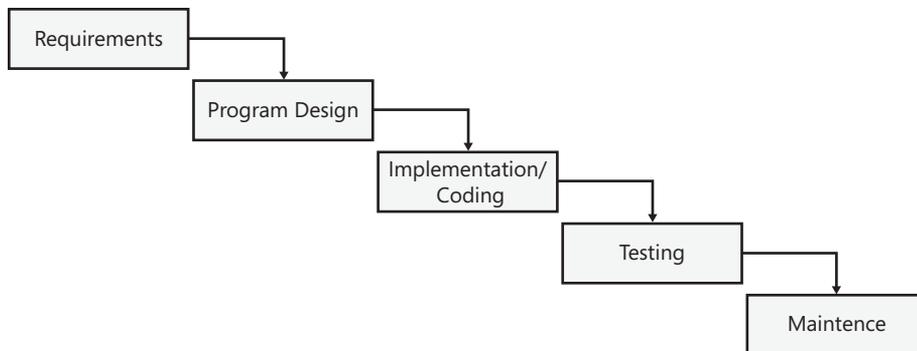


FIGURE 3-1 Waterfall model.

The advantage of this model is that when you begin a phase, everything from the previous phase is complete. Design, for example, will never begin before the requirements are complete. Another potential benefit is that the model forces you to think and design as much as possible before beginning to write code. Taken literally, waterfall is inflexible because it doesn't appear to allow phases to repeat. If testing, for example, finds a bug that leads back to a design flaw, what do you do? The Design phase is "done." This apparent inflexibility has led to many criticisms of waterfall. Each stage has the potential to delay the entire product cycle, and in a long product cycle, there is a good chance that at least some parts of the early design become irrelevant during implementation.

An interesting point about waterfall is that the inventor, Winston Royce, intended for waterfall to be an iterative process. Royce's original paper on the model¹ discusses the need to iterate at least twice and use the information learned during the early iterations to influence later iterations. Waterfall was invented to improve on the stage-based model in use for decades by recognizing feedback loops between stages and providing guidelines to minimize the impact of rework. Nevertheless, waterfall has become somewhat of a ridiculed process among many software engineers—especially among Agile proponents. In many circles of software engineering, *waterfall* is a term used to describe *any* engineering system with strict processes.

Spiral Model

In 1988, Barry Boehm proposed the spiral model of software development.² Spiral, as shown in Figure 3-2, is an iterative process containing four main phases: determining objectives, risk evaluation, engineering, and planning for the next iteration.

- **Determining objectives** Identify and set specific objectives for the current phase of the project.
- **Risk evaluation** Identify key risks, and identify risk reduction and contingency plans. Risks might include cost overruns or resource issues.
- **Engineering** In the engineering phase, the work (requirements, design, development, testing, and so forth) occurs.
- **Planning** The project is reviewed, and plans for the next round of the spiral begin.

¹ Winston Royce, "Managing the Development of Large Software Systems," *Proceedings of IEEE WESCON 26* (August 1970).

² Barry Boehm, "A Spiral Model of Software Development," *IEEE* 21, no. 5 (May 1988): 61–72.

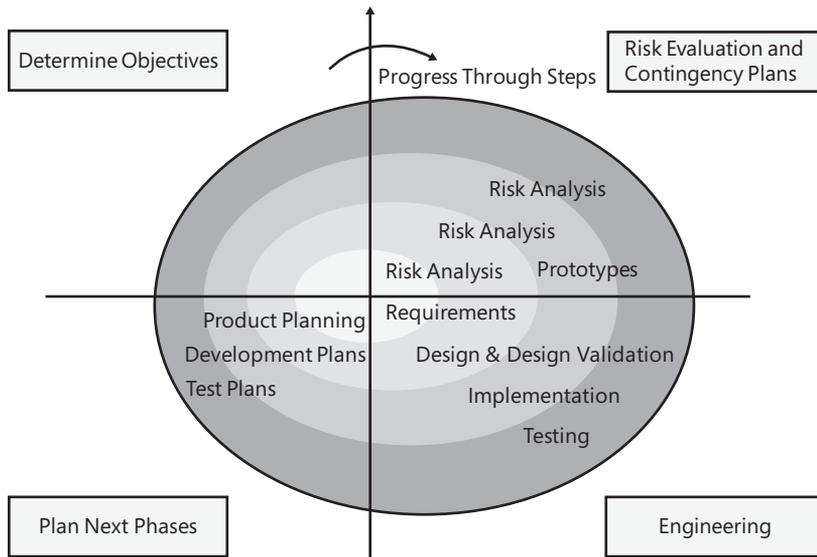


FIGURE 3-2 Simplified spiral model.

Another important concept in the spiral model is the repeated use of prototypes as a means of minimizing risk. An initial prototype is constructed based on preliminary design and approximates the characteristics of the final product. In subsequent iterations, the prototypes help evaluate strengths, weaknesses, and risks.

Software development teams can implement spiral by initially planning, designing, and creating a bare-bones or prototype version of their product. The team then gathers customer feedback on the work completed, and then analyzes the data to evaluate risk and determine what to work on in the next iteration of the spiral. This process continues until either the product is complete or the risk analysis shows that scrapping the project is the better (or less risky) choice.

Agile Methodologies

By using the spiral model, teams can build software iteratively—building on the successes (and failures) of the previous iterations. The planning and risk evaluation aspects of spiral are essential for many large software products but are too process heavy for the needs of many software projects. Somewhat in response to strict models such as waterfall, Agile approaches focus on lightweight and incremental development methods.

Agile methodologies are currently quite popular in the software engineering community. Many distinct approaches fall under the Agile umbrella, but most share the following traits:

- **Multiple, short iterations** Agile teams strive to deliver working software frequently and have a record of accomplishing this.
- **Emphasis on face-to-face communication and collaboration** Agile teams value interaction with each other and their customers.
- **Adaptability to changing requirements** Agile teams are flexible and adept in dealing with changes in customer requirements at any point in the development cycle. Short iterations allow them to prioritize and address changes frequently.
- **Quality ownership throughout the product cycle** Unit testing is prevalent among developers on Agile teams, and many use test-driven development (TDD), a method of unit testing where the developer writes a test before implementing the functionality that will make it pass.

In software development, to be Agile means that teams can quickly change direction when needed. The goal of always having working software by doing just a little work at a time can achieve great results, and engineering teams can almost always know the status of the product. Conversely, I can recall a project where we were “95 percent complete” for at least three months straight. In hindsight, we had no idea how much work we had left to do because we tried to do everything at once and went months without delivering working software. The goal of Agile is to do a little at a time rather than everything at once.

Other Models

Dozens of models of software development exist, and many more models and variations will continue to be popular. There isn’t a best model, but understanding the model and creating software within the bounds of whatever model you choose can give you a better chance of creating a quality product.

Milestones

It’s unclear if it was intentional, but most of the Microsoft products I have been involved in used the spiral model or variations.³ When I joined the Windows 95 team at Microsoft, they were in the early stages of “Milestone 8” (or M8 as we called it). M8, like one of its predecessors, M6, ended up being a public beta. Each milestone had specific goals for product functionality and quality. Every product I’ve worked on at Microsoft, and many others I’ve worked with indirectly, have used a milestone model.

³ Since I left product development in 2005 to join the Engineering Excellence team, many teams have begun to adopt Agile approaches.

The milestone schedule establishes the time line for the project release and includes key interim project deliverables and midcycle releases (such as beta and partner releases). The milestone schedule helps individual teams understand the overall project expectations and to check the status of the project. An example of the milestone approach is shown in Figure 3-3.



FIGURE 3-3 Milestone model example.

The powerful part of the milestone model is that it isn't just a date drawn on the calendar. For a milestone to be complete, specific, predefined criteria must be satisfied. The criteria typically include items such as the following:

- **"Code complete" on key functionality** Although not completely tested, the functionality is implemented.
- **Interim test goals accomplished** For example, code coverage goals or tests completed goals are accomplished.
- **Bug goals met** For example, no severity 1 bugs or no crashing bugs are known.
- **Nonfunctional goals met** For example, performance, stress, load testing is complete with no serious issues.

The criteria usually grow stricter with each milestone until the team reaches the goals required for final release. Table 3-1 shows the various milestones used in a sample milestone project.

TABLE 3-1 Example Milestone Exit Criteria (partial list)

Area	Milestone 1	Milestone 2	Milestone 3	Release
Test case execution		All Priority 1 test cases run	All Priority 1 and 2 test cases run	All test cases run
Code coverage	Code coverage measured and reports available	65% code coverage	75% code coverage	80% code coverage
Reliability	Priority 1 stress tests running nightly	Full stress suite running nightly on at least 200 computers	Full stress suite running nightly on at least 500 computers with no uninvestigated issues	Full stress suite running nightly on at least 500 computers with no uninvestigated issues
Reliability		Fix the top 50% of customer-reported crashes from M1	Fix the top 60% of customer-reported crashes from M2	Fix the top 70% of customer-reported crashes from M3

Area	Milestone 1	Milestone 2	Milestone 3	Release
Features		New UI shell in 20% of product	New UI in 50% of product and usability tests complete	New UI in 100% of product and usability feedback implemented
Performance	Performance plan, including scalability goals, complete	Performance baselines established for all primary customer scenarios	Full performance suite in place with progress tracking toward ship goals	All performance tests passing, and performance goals met

Another advantage of the milestone model (or any iterative approach) is that with each milestone, the team gains some experience going through the steps of release. They learn how to deal with surprises, how to ask good questions about unmet criteria points, and how to anticipate and handle the rate of incoming bugs. An additional intent is that each milestone release functions as a complete product that can be used for large-scale testing (even if the milestone release is not an external beta release). Each milestone release is a complete version of the product that the product team and any other team at Microsoft can use to “kick the tires” on (even if the tires are made of cardboard).

The quality milestone

Several years ago, I was on a product team in the midst of a ship cycle. I was part of the daily bug triage, where we reviewed, assigned, and sometimes postponed bugs to the next release. Postponements happen for a variety of reasons and are a necessary part of shipping software. A few months before shipping, we had some time left at the end of the meeting, and I asked if we could take a quick look at the bugs assigned to the next version of our product. The number was astounding. It was so large that we started calling it the “wave.” The wave meant that after we shipped, we would be starting work on the next release with a huge backlog of product bugs.

Bug backlog along with incomplete documents and flaky tests we need to fix “someday” are all items that add up to *technical debt*.⁴ We constantly have to make tradeoffs when developing software, and many of those tradeoffs result in technical debt. Technical debt is difficult to deal with, but it just doesn’t go away if we ignore it, so we have to do something. Often, we try to deal with it while working on other things or in the rare times when we get a bit of a lull in our schedules. This is about as effective as bailing out a leaky boat with a leaky bucket.

Another way many Microsoft teams have been dealing with technical debt is with a *quality milestone*, or MQ. This milestone, which occurs after product release but before

⁴ Matthew Heusser writes about technical debt often on his blog (xndev.blogspot.com). Matt doesn’t work for Microsoft...yet.

getting started on the next wave of product development, provides an opportunity for teams to fix bugs, retool their infrastructure, and fix anything else pushed aside during the previous drive to release. MQ is also an opportunity to implement improvements to any of the engineering systems or to begin developing early prototypes of work and generate new ideas.

Beginning a product cycle with the backlog of bugs eliminated, the test infrastructure in place, improvement policies implemented, and everything else that annoyed you during the previous release resolved is a great way to start work on a new version of a mature product.

Agile at Microsoft

Agile methodologies are popular at Microsoft. An internal e-mail distribution list dedicated to discussion of Agile methodologies has more than 1,500 members. In a survey sent to more than 3,000 testers and developers at Microsoft, approximately one-third of the respondents stated that they used some form of Agile software development.⁵

Feature Crews

Most Agile experts state that a team size of 10 or less collocated team members is optimal. This is a challenge for large-scale teams with thousands or more developers. A solution commonly used at Microsoft to scale Agile practices to large teams is the use of *feature crews*.

A feature crew is a small, cross-functional group, composed of 3 to 10 individuals from different disciplines (usually Dev, Test, and PM), who work autonomously on the end-to-end delivery of a functional piece of the overall system. The team structure is typically a program manager, three to five testers, and three to five developers. They work together in short iterations to design, implement, test, and integrate the feature into the overall product, as shown in Figure 3-4.

The key elements of the team are the following:

- It is independent enough to define its own approach and methods.
- It can drive a component from definition, development, testing, and integration to a point that shows value to the customer.

Teams in Office and Windows use this approach as a way to enable more ownership, more independence, and still manage the overall ship schedule. For the Office 2007 project, there were more than 3,000 feature crews.

⁵ Nachiappan Nagappan and Andrew Begel, "Usage and Perceptions of Agile Software Development in an Industrial Context: An Exploratory Study," 2007, <http://csdl2.computer.org/persagen/DLabsToc.jsp?resourcePath=/dl/proceedings/&toc=comp/proceedings/esem/2007/2886/00/2886toc.xml&DOI=10.1109/ESEM.2007.85>.

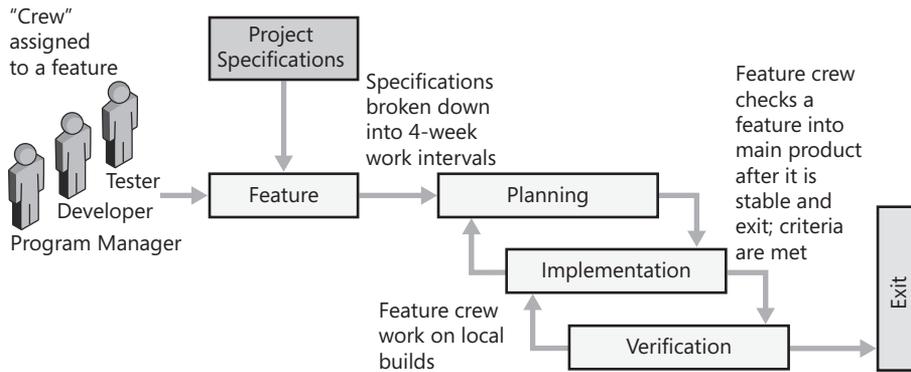


FIGURE 3-4 Feature crew model.

Getting to Done

To deliver high-quality features at the end of each iteration, feature crews concentrate on defining “done” and delivering on that definition. This is most commonly accomplished by defining *quality gates* for the team that ensure that features are complete and that there is little risk of feature integration causing negative issues. Quality gates are similar to milestone exit criteria. They are critical and often require a significant amount of work to satisfy. Table 3-2 lists sample feature crew quality gates.⁶

TABLE 3-2 Sample Feature Crew Quality Gates

Quality gate	Description
Testing	All planned automated tests and manual tests are completed and passing.
Feature Bugs Closed	All known bugs found in the feature are fixed or closed.
Performance	Performance goals for the product are met by the new feature.
Test Plan	A test plan is written that documents all planned automated and manual tests.
Code Review	Any new code is reviewed to ensure that it meets code design guidelines.
Functional Specification	A functional spec has been completed and approved by the crew.
Documentation Plan	A plan is in place for the documentation of the feature.
Security	Threat model for the feature has been written and possible security issues mitigated.
Code Coverage	Unit tests for the new code are in place and ensure 80% code coverage of the new feature.
Localization	The feature is verified to work in multiple languages.

⁶ This table is based on Ade Miller and Eric Carter, “Agile and the Inconceivably Large,” *IEEE* (2007).

The feature crew writes the necessary code, publishes private releases, tests, and iterates while the issues are fresh. When the team meets the goals of the quality gates, they migrate their code to the main product source branch and move on to the next feature. I. M. Wright's *Hard Code* (Microsoft Press, 2008) contains more discussion on the feature crews at Microsoft.

Iterations and Milestones

Agile iterations don't entirely replace the milestone model prevalent at Microsoft. Agile practices work hand in hand with milestones—on large product teams, milestones are the perfect opportunity to ensure that all teams can integrate their features and come together to create a product. Although the goal on Agile teams is to have a shippable product at all times, most Microsoft teams release to beta users and other early adopters every few months. Beta and other early releases are almost always aligned to product milestones.

Putting It All Together

At the micro level, the smallest unit of output from developers is code. Code grows into functionality, and functionality grows into features. (At some point in this process, test becomes part of the picture to deliver *quality* functionality and features.)

In many cases, a large group of features becomes a *project*. A project has a distinct beginning and end as well as checkpoints (milestones) along the way, usage scenarios, personas, and many other items. Finally, at the top level, subsequent releases of related projects can become a product line. For example, Microsoft Windows is a product line, the Windows Vista operating system is a project within that product line, and hundreds of features make up that project.

Scheduling and planning occur at every level of output, but with different context, as shown in Figure 3-5. At the product level, planning is heavily based on long-term strategy and business need. At the feature level, on the other hand, planning is almost purely tactical—getting the work done in an effective and efficient manner is the goal. At the project level, plans are often both tactical and strategic—for example, integration of features into a scenario might be tactical work, whereas determining the length of the milestones and what work happens when is more strategic. Classifying the work into these two buckets isn't important, but it is critical to integrate strategy and execution into large-scale plans.

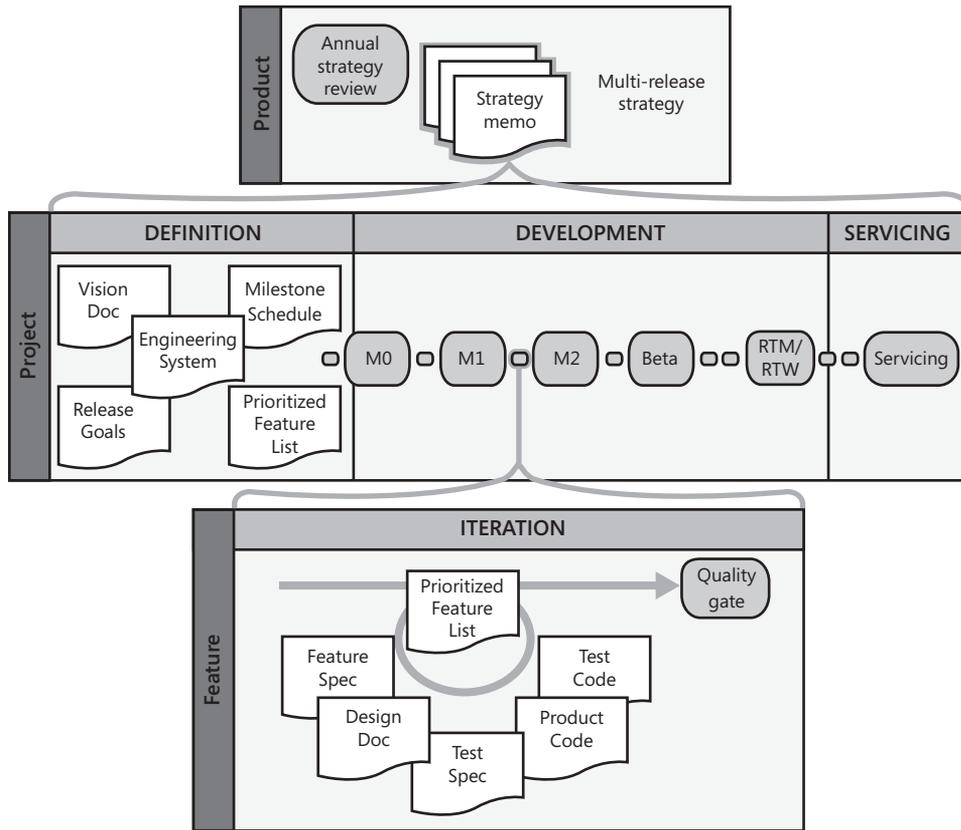


FIGURE 3-5 Software life cycle workflow.

Process Improvement

In just about anything I take seriously, I want to improve continuously. Whether I'm preparing a meal, working on my soccer skills, or practicing a clarinet sonata, I want to get better. Good software teams have the same goal—they reflect often on what they're doing and think of ways to improve.

Dr. W. Edwards Deming is widely acknowledged for his work in quality and process improvement. One of his most well known contributions to quality improvement was the simple *Plan, Do, Check, Act* cycle (sometimes referred to as the Shewhart cycle, or the PDCA cycle). The following phases of the PDCA cycle are shown in Figure 3-6:

- **Plan** Plan ahead, analyze, establish processes, and predict the results.
- **Do** Execute on the plan and processes.

- **Check** Analyze the results (note that Deming later changed the name of this stage to “Study” to be more clear).
- **Act** Review all steps and take action to improve the process.

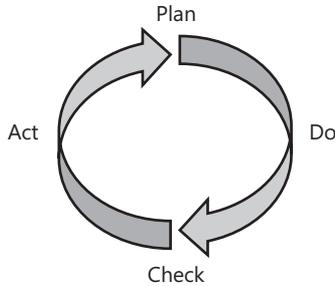


FIGURE 3-6 Deming's PDCA cycle.

For many people, the cycle seems so simple that they see it as not much more than common sense. Regardless, this is a powerful model because of its simplicity. The model is the basis of the Six Sigma DMAIC (Define, Measure, Analyze, Improve, Control) model, the ADDIE (Analyze, Design, Develop, Implement, Evaluate) instructional design model, and many other improvement models from a variety of industries.

Numerous examples of applications of this model can be found in software. For example, consider a team who noticed that many of the bugs found by testers during the last milestone could have been found during code review.

1. First, the team plans a process around code reviews—perhaps requiring peer code review for all code changes. They also might perform some deeper analysis on the bugs and come up with an accurate measure of how many of the bugs found during the previous milestone could potentially have been found through code review.
2. The group then performs code reviews during the next milestone.
3. Over the course of the next milestone, the group monitors the relevant bug metrics.
4. Finally, they review the entire process, metrics, and results and determine whether they need to make any changes to improve the overall process.

Formal Process Improvement Systems at Microsoft

Process improvement programs are prevalent in the software industry. ISO 9000, Six Sigma, Capability Maturity Model Integrated (CMMI), Lean, and many other initiatives all exist to help organizations improve and meet new goals and objectives. The different programs all focus on process improvement, but details and implementation vary slightly. Table 3-3 briefly describes some of these programs.

TABLE 3-3 Formal Process Improvement Programs

Process	Concept
ISO 9000	A system focused on achieving customer satisfaction through satisfying quality requirements, monitoring processes, and achieving continuous improvement.
Six Sigma	Developed by Motorola. Uses statistical tools and the DMAIC (Define, Measure, Analyze, Implement, Control) process to measure and improves processes.
CMMI	Five-level maturity model focused on project management, software engineering, and process management practices. CMMI focuses on the organization rather than the project.
Lean	Focuses on eliminating waste (for example, defects, delay, and unnecessary work) from the engineering process.

Although Microsoft hasn't wholeheartedly adopted any of these programs for widespread use, process improvement (either formal or ad hoc) is still commonplace. Microsoft continues to take process improvement programs seriously and often will "test" programs to get a better understanding of how the process would work on Microsoft products. For example, Microsoft has piloted several projects over the past few years using approaches based on Six Sigma and Lean. The strategy in using these approaches to greatest advantage is to understand how best to achieve a balance between the desire for quick results and the rigor of Lean and Six Sigma.

Microsoft and ISO 9000

Companies that are ISO 9000 certified have proved to an auditor that their processes and their adherence to those processes are conformant to the ISO standards. This certification can give customers a sense of protection or confidence in knowing that quality processes were integral in the development of the product.

At Microsoft, we have seen customers ask about our conformance to ISO quality standards because generally they want to know if we uphold quality standards that adhere to the ISO expectations in the development of our products.

Our response to questions such as this is that our development process, the documentation of our steps along the way, the support our management team has for quality processes, and the institutionalization of our development process in documented and repeatable processes (as well as document results) are all elements of the core ISO standards and that, in most cases, we meet or exceed these.

This doesn't mean, of course, that Microsoft doesn't value ISO 9000, and neither does it mean that Microsoft will never have ISO 9000-certified products. What it does mean at the time of this writing is that in most cases we feel our processes and standards fit the needs of our engineers and customers as well as ISO 9000 would. Of course, that could change next week, too.

Shipping Software from the War Room

Whether it's the short product cycle of a Web service or the multiyear product cycle of Windows or Office, at some point, the software needs to ship and be available for customers to use. The decisions that must be made to determine whether a product is ready to release, as well as the decisions and analysis to ensure that the product is on the right track, occur in the *war room* or *ship room*. The war team meets throughout the product cycle and acts as a ship-quality oversight committee. As a name, "war team" has stuck for many years—the term describes what goes on in the meeting: "conflict between opposing forces or principles."

As the group making the day-to-day decisions for the product, the war team needs a holistic view of all components and systems in the entire product. Determining which bugs get fixed, which features get cut, which parts of the team need more resources, or whether to move the release date are all critical decisions with potentially serious repercussions that the war team is responsible for making.

Typically, the war team is made up of one representative (usually a manager) from each area of the product. If the representative is not able to attend, that person nominates someone from his or her team to attend instead so that consistent decision making and stakeholder buy-in can occur, especially for items considered plan-of-record for the project.

The frequency of war team meetings can vary from once a week during the earliest part of the ship cycle to daily, or even two or three times a day in the days leading up to ship day.

War, What Is It Good For?

The war room is the pulse of the product team. If the war team is effective, everyone on the team remains focused on accomplishing the right work and understands why and how decisions are made. If the war team is unorganized or inefficient, the pulse of the team is also weak—causing the myriad of problems that come with lack of direction and poor leadership.

Some considerations that lead to a successful war team and war room meetings are the following:

- Ensure that the right people are in the room. Missing representation is bad, but too many people can be just as bad.
- Don't try to solve every problem in the meeting. If an issue comes up that needs more investigation, assign it to someone for follow-up and move on.
- Clearly identify action items, owners, and due dates.
- Have clear issue tracking—and address issues consistently. Over time, people will anticipate the flow and be more prepared.

- Be clear about what you want. Most ship rooms are focused and crisp. Some want to be more collaborative. Make sure everyone is on the same page. If you want it short and sweet, don't let discussions go into design questions, and if it's more informal, don't try to cut people off.
- Focus on the facts rather than speculation. Words like "I think," "It might," "It could" are red flags. Status is like pregnancy—you either are or you aren't; there's no in between.
- Everyone's voice is important. A phrase heard in many war rooms is "Don't listen to the HiPPO"—where HiPPO is an acronym for highest-paid person's opinion.
- Set up exit criteria in advance at the beginning of the milestone, and hold to them. Set the expectation that quality goals are to be adhered to.
- One person runs the meeting and keeps it moving in an orderly manner.
- It's OK to have fun.

Defining the Release—Microspeak

Much of the terminology used in the room might confuse an observer in a ship team meeting. Random phrases and three-letter acronyms (TLAs) flow throughout the conversation. Some of the most commonly used terms include the following:

- **LKG** "Last Known Good" release that meets a specific quality bar. Typically, this is similar to self-host.
- **Self-host** A self-host build is one that is of sufficient quality to be used for day-to-day work. The Windows team, for example, uses internal prerelease versions of Windows throughout the product cycle.
- **Self-toast** This is a build that completely ruins, or "toasts," your ability to do day-to-day work. Also known as *self-hosed*.
- **Self-test** A build of the product that works well enough for most testing but has one or more blocking issues keeping it from reaching self-host status.
- **Visual freeze** Point or milestone in product development cycle when visual/UI changes are locked and will not change before release.
- **Debug/checked build** A build with a number of features that facilitate debugging and testing enabled.
- **Release/free build** A build optimized for release.
- **Alpha release** A very early release of a product to get preliminary feedback about the feature set and usability.
- **Beta release** A prerelease version of a product that is sent to customers and partners for evaluation and feedback.

Mandatory Practices

Microsoft executive management doesn't dictate how divisions, groups, or teams develop and test software. Teams are free to experiment, use tried-and-true techniques, or a combination of both. They are also free to create their own mandatory practices on the team or division level as the context dictates. Office, for example, has several criteria that every part of Office must satisfy to ship, but those same criteria might not make sense in a small team shipping a Web service. The freedom in development processes enables teams to innovate in product development and make their own choices. There are, however, a select few required practices and policies that every team at Microsoft must follow.

These mandatory requirements have little to do with the details of shipping software. The policies are about making sure that several critical steps are complete prior to shipping a product.

There are few mandatory engineering policies, but products that fail to adhere to these policies are not allowed to ship. Some examples of areas included in mandatory policies include planning for privacy issues, licenses for third-party components, geopolitical review, virus scanning, and security review.

Expected vs. Mandatory

Mandatory practices, if not done in a consistent and systematic way, create unacceptable risk to customers and Microsoft.

Expected practices are effective practices that every product group should use (unless there is a technical limitation). The biggest example of this is the use of static analysis tools. (See Chapter 11, "Non-functional Testing.") When we first developed C#, for example, we did not have static code analysis tools for that language. It wasn't long after the language shipped, however, before teams developed static analysis tools for C#.

One-Stop Shopping

Usually, one person on a product team is responsible for release management. Included in that person's duties is the task of making sure all of the mandatory obligations have been met. To ensure that everyone understands mandatory policies and applies them consistently, every policy, along with associated tools and detailed explanations, is located on a single internal Web portal so that Microsoft can keep the number of mandatory policies as low as possible and supply a consistent toolset for teams to satisfy the requirements with as little pain as possible.

Summary: Completing the Meal

Like creating a meal, there is much to consider when creating software—especially as the meal (or software) grows in size and complexity. Add to that the possibility of creating a menu for an entire week—or multiple releases of a software program—and the list of factors to consider can quickly grow enormous.

Considering *how* software is made can give great insights into what, where, and when the “ingredients” of software need to be added to the application soup that software engineering teams put together. A plan, recipe, or menu can help in many situations, but as Eisenhower said, “In preparing for battle I have always found that plans are useless, but planning is indispensable.” The point to remember is that putting some effort into thinking through everything from the implementation details to the vision of the product can help achieve results. There isn’t a *best* way to make software, but there are several *good* ways. The good teams I’ve worked with don’t worry nearly as much about the actual process as they do about successfully executing whatever process they are using.

Index

A

- Abort result (automated tests), 242
- access (external) to bug tracking system, 191
- accessibility testing, 265–268. *See also* usability testing
- Accessible Event Watcher (AccEvent) tool, 268
- Accessible Explorer tool, 268
- accountability for quality, 368
- accuracy of test automation, 221
- Act phase (PDCA cycle), 52
- action simulation, in UI automation, 224
- active (bug status), 193
- Active Accessibility software development kit, 268
- activities of code reviews, monitoring, 382
- ad hoc approach, combinatorial testing, 101
- ADDIE model, 52
- “Adopt an App” program, 261
- AFA (automatic failure analysis), 365–371
- Agile methodologies, 44, 48–50
- all states path (graph theory), 167
- all transitions path (graph theory), 167
- alpha release, defined, 55
- analysis (test pattern attribute), 63
- analysis phase (SEARCH test automation), 229, 240–242
- analysis tools. *See* static analysis
- analytical problem solving competency, 28
- anticipating. *See* planning
- API testing with models, 168, 171
- application compatibility testing, 261–263
- application libraries, 261, 385
- application usage data, collecting, 300
- Application Verifier tool, 262
- approving big fixes. *See* triage for managing bugs
- asking questions
 - performance measurement, 253–255
 - test design, 65
- assignment of bugs
 - automated notification of, 191
 - limits on, 205–209
 - specified in bug reports, 193
- assumptions about test case readers, 211

- audio features, accessibility of, 268
- authentication with WLID, 334
- Automate Everything attribute (BVTs), 283
- automated service deployment, 337
- automated tests, 213. *See also* test automation
 - as invalid testing solution, 231
 - test code analysis, 292
 - tracking changes in, 275
- automatic failure analysis (AFA), 365–371
- automating models, 166–171
- automation (test case attribute), 213
- AutomationElement elements, 226
- awareness, interpersonal (competency), 29

B

- backlog of bugs, 47
- Ball, Tom, 36
- Ballmer, Steve, 3
- base choice (BC) approach, combinatorial testing, 102
 - insufficiency of, 111
- baseline performance, establishing, 253
- basic control flow diagrams (CFDs), 122
- basis path testing, 117–142
- BATs (build acceptance tests), 283
- Bayesian Graphical Modeling (BGM), 172
- BC (base choice) approach, combinatorial testing, 102
 - insufficiency of, 111
- behavioral testing. *See* non-functional testing
- Beizer, Boris, 75
- Bergman, Mark, 229
- best guess approach, combinatorial testing, 101
- beta release
 - defined, 55
 - identifying product as, 336
- BGM (Bayesian Graphical Modeling), 172
- bias in white box testing, 116
- Big Challenges (company value), 4
- big company, Microsoft as, 7
 - working small in a big company, 11–14
- big-picture performance considerations, 253
- Binder, Robert, 29, 62

- BIOS clocks, 85
- black box testing, 71
- Block result (automated tests), 242, 243
- block testing, 118–126
- Boehm, Barry, 43
- bottlenecks, anticipating, 254. *See also* performance testing
- boundary condition tests, 202
- boundary value analysis (BVA), 90–100
 - defining boundary values, 90
 - hidden boundaries, 97
 - using ECP tables with, 88, 93
- Boundary Value Analysis Test Pattern (example), 63
- branching control flow, testing. *See* decision testing
- breadth of Microsoft portfolio, 7
- breaking builds, 284
- broken window theory, 289
- browser-based service performance metrics, 351–356
- brute-force UI automation, 227
- buckets for errors, 307, 358
- bug backlog, 47
- bug bars (limits on bugs assigned), 205–209
- bug lifecycle, 189–190
- bug metrics, 201
 - churn metrics and, 274
- bug morphing, 199
- bug notification system, 191
- bug reports, 188, 192–197
 - examples of, 190
- bug type (bug report field), 195
- bug workflow, 188
- bugs. *See* managing bugs
- bugs, specific
 - “Adopt an App” program, 261
 - “disallow server cascading” failure, 361
 - Friday the 13th bug, 85
 - love bug, 208
 - milk carton bug, 207
 - tsunami effect, 361
 - USB cart of death, 257
- bugs vs. features, 197
- build acceptance tests (BATs), 283
- build labs, 282
- build process, 281–287
 - breaking builds, 284
 - testing daily builds, 374
- build verification tests (BVTs), 283

- building services on servers, 331
- built-in accessibility features, 265
- BUMs (business unit managers), 12
- Business Division. *See* MBD
- business goals, in test time estimation, 65
- business unit managers. *See* BUMs
- BVA (boundary value analysis), 90–100
 - defining boundary values, 90
 - hidden boundaries, 97
 - using ECP tables with, 88, 93
- BVA Test Pattern (example), 63
- BVTs (build verification tests), 283
- by design (bug resolution value), 195, 197

C

- C# development, 236
- calculating code complexity. *See* code complexity
- call center data, collecting, 357
- campus recruiting, 29–31
- capacity testing, 256
- Carbon Allocation Model (CarBAM), 324
- career stages, 33
- careers at Microsoft, 33–38
 - recruiting. *See* recruiting testers
 - in Test, 34–38
- Catlett, David, 68
- CBO metric, 154
- CEIP (Customer Experience Improvement Plan), 299–304
- CER (Corporate Error Reporting), 307
- CFDs (control flow diagrams), 122
- chair of test leadership team, 370
- changes in code
 - number of (code churn), 273–275
 - tracking. *See* source control
- Check phase (PDCA cycle), 52
- check-in systems, 286
- checklists for code reviews, 381
- churn, 273–275
- CIS (Cloud Infrastructure Services), 320
- CK metrics, 153
- class-based complexity metrics, 153
- classic Microsoft bugs, 207–209
- ClassSetup attribute, 237
- ClassTeardown attribute, 237
- clean machines, 254
- cleanup phase (SEARCH test automation), 229, 243

- closed (bug status), 193
- cloud services, 320
- CLR (Common Language Runtime), 179
- CMMI (Capability Maturity Model Integrated), 53
- code analysis. *See* static analysis
- code churn, 273–275
- code complexity
 - in automated tests, 247
 - cyclomatic complexity, 133
 - measuring, 63, 149–152, 155
 - estimating (code smell), 147
 - estimating test time, 65
 - Halstead metrics, 152–153
 - how to use metrics for, 157
 - lines of code (LOC), 147–148
 - object-oriented metrics, 153
 - quantifying, 146–148
 - risk from, 145–158
- code coverage
 - analysis tools for, 293
 - behavioral and exploratory testing, 117
 - combinatorial analysis and, 112
 - with functional testing, 77
 - milestone criteria, 46
 - quality gates, 49
 - scripted tests, 117
 - statement vs. block coverage, 118
- code reuse, 385–387
- code reviews, 379–384
 - collateral data with, 384
 - measuring effectiveness of, 381–384
- code smell, 147
- code snapshots, 275, 374, 378
- CodeBox portal, 386
- Cole, David, 261
- collaboration. *See* communication
- collecting data from customers. *See* CEIP (Customer Experience Improvement Plan)
- combination tests, 102
- combinatorial analysis
 - effectiveness of, 111
 - tools for, in practice, 104–111
- communication, 380
 - cross-boundary collaboration (competency), 28
- company values at Microsoft, 4
- comparing documents, 276
- compatibility testing, 261–263
 - dogfooding (being users), 264, 350
- competencies, 27
- compilation errors, 284, 285
- complexity of code
 - in automated tests, 247
 - cyclomatic complexity, 133
 - measuring, 63, 149–152, 155
 - estimating (code smell), 147
 - estimating test time, 65
 - Halstead metrics, 152–153
 - how to use metrics for, 157
 - lines of code (LOC), 147–148
 - object-oriented metrics, 153
 - quantifying, 146–148
 - risk from, 145–158
- complexity of test automation, 221
- compound conditional clauses, testing, 129
- compressibility (metric), 352, 353
- computer-assisted testing, 230
- computing innovations, waves of, 329
- condition testing, 129–132
- conditional clauses, testing. *See* condition testing; decision testing
- conditions (test case attribute), 212
- confidence (competency), 28
 - with functional testing, 77
- configurability of bug tracking system, 191
- configuration data, collecting, 300
- configurations (test case attribute), 212
- conformance, in test time estimation, 65
- Connect site, 312
- container-based datacenters (container SKUs), 322
- Content discipline, 15
- context, bug, 194
- continuous improvement. *See* process improvement
- contrast, display, 268
- control flow diagrams (CFDs), 122
- control flow graphs, 149
- control flow modeling, 118, 122
- control flow testing. *See* structural testing
- control testability, 67
- Corporate Error Reporting (CER), 307
- cost of quality, 369
- cost of test automation, 220
- count of changes (churn metric), 273
- counters (performance), 254
- counting bugs, 200, 204
- counting test cases, 215–216, 217
- coupling, services, 333–335, 346

coupling between object classes (metric), 154
 coverage method (WER), 308
 Creative discipline, 16
 credit card processing, 335
 criteria for milestones, 46
 quality gates as, 49
 Critical attribute (BVTs), 283
 cross-boundary collaboration (competency), 28
 cross-referencing test cases with automation
 scripts, 246
 culture of quality, 366
 Customer Experience Improvement Plan (CEIP),
 299–304
 customer feedback systems, 297–315
 connecting with customers, 312–315
 emotional response, 309–312
 for services, 357
 testing vs. quality, 297–299
 watching customers. *See* CEIP (Customer
 Experience Improvement Plan)
 Windows Error Reporting, 304–309
 customer impact of bugs, 194
 customer-driven testing, 303
 customer-focused innovation, 28
 Cutter, David, 33
 cyclomatic complexity, 133
 measuring, 149–152, 155
 practical interpretations, 63
 Czerwonka, Jacek, 111

D

daily builds, 281–287
 testing with virtualization, 374
 data coverage. *See* code coverage; equivalence
 class partitioning (ECP)
 data sanitization, 351
 DDE (defect detection effectiveness), 111
 debug/checked builds, 55
 Debuggable and Maintainable attribute (BVTs),
 283
 debuggers, exploratory testing with, 66
 debugging after automated tests, 247
 debugging scope, 277
 decision testing, 126–129. *See also* condition
 testing
 decisions in programs, counting. *See* cyclomatic
 complexity, measuring
 decomposing variable data (ECP), 80–82

dedicated teams for non-functional testing,
 251
 defect detection effectiveness (DDE), 111
 defect removal efficiency (DRE), 191
 Deming, W. Edwards, 51
 dependencies, services, 333
 dependency errors, 285
 deploying services with automation, 337
 deployment test clusters (services), 344, 345
 depth of inheritance tree (metric), 153
 descriptions for test patterns, 63
 descriptions of bugs (in bug reports), 192
 design (test pattern attribute), 63
 design, importance of, 61
 design patterns, 62, 309
 designing models, 161
 finite state models, 166
 designing test cases, 61–72
 best practices, 61
 estimating test time, 64
 getting started, 65–67
 practical considerations, 70–72
 testability, 67–69
 testing good and bad, 69
 using test patterns, 62–64
 DeVaan, Jon, 33
 Development (SDE) discipline, 15
 development models, 42–45
 devices. *See* hardware
 dijzjzjz½ vu heuristic, 99
 Difficulty metric (Halstead), 152
 diff utilities, 276
 Director, Software Development Engineer in
 Test title, 37
 Director of Test, 38
 Director of Test Excellence, 381
 “disallow server cascading” failure, 361
 disciplines, product engineering, 15, 21
 display contrast (accessibility), 268
 distributed stress testing, 257
 DIT metric, 153
 diversity of Microsoft portfolio, 7
 divisions at Microsoft, 5
 DMAIC model, 52
 Do phase (PDCA cycle), 51
 document comparison tools, 276
 documentation of test cases. *See* entries at log;
 test cases
 documenting code changes. *See* source control
 dogfooding, 264, 350

“done,” defining, 49
 down-level browser experience, 326
 DRE (defect removal efficiency), 191
 Drotter, Stephen, 33
 duplicate (bug resolution value), 195
 duplicate bugs, 200

E

E&D (Entertainment and Devices Division), 5
 each choice (EC) approach, combinatorial testing, 102
 ease of use, bug tracking system, 190
 “eating our dogfood”, 264, 350
 ECP (equivalence class partitioning), 78–90
 analyzing parameter subsets, 84–86
 boundary condition tests, 88
 boundary value analysis with, 88, 93
 decomposing variable data, 80–82
 example of, 83
 edges (control node graphs), 150
 education strategy, 66, 67
 EE (Engineering Excellence) group, 32
 effectiveness of code reviews, measuring, 381–384
 effort of test automation, determining, 220
 80:20 rule, 145
 Elop, Stephen, 5
 e-mail discussions in bug reports, 199
 emotional response from customers, 309–312
 employee orientation, 32
 emulating services, 346
 ending state (model), 160
 engineering career at Microsoft, 33
 engineering disciplines, 15, 21
 Engineering Excellence (EE) group, 32
 Engineering Excellence (EE) team, 378
 Engineering Excellence Forum, 379, 381
 engineering life cycles, 41–57
 Agile methodologies, 44, 48–50
 milestones, 45–48
 process improvement, 51–53
 formal systems for, 52
 shipping software, 54–56
 software development models, 42–45
 Engineering Management discipline, 16
 engineering organizational models, 8
 engineering workforce (Microsoft), size of, 27
 campus recruiting, 29–31
 engineers, types of, 14–17
 Entertainment and Devices Division. *See* E&D

environment, bug, 194
 environmental sensitivity of automated tests, 292
 equivalence class partitioning (ECP), 78–90
 analyzing parameter subsets, 84–86
 boundary condition tests, 88
 boundary value analysis with, 88, 93
 decomposing variable data, 80–82
 example of, 83
 estimation of test time, 64
 ET. *See* exploratory testing
 Euler, Leonhard, 166
 examples for test patterns, 63
 exception handling, block testing for, 124
 execution phase (SEARCH test automation), 229, 233–240
 exit criteria for milestones, 46
 quality gates as, 49
 expected practices, 56
 experience quality, 298
 expiration date set (metric), 352, 354
 exploratory testing, 116
 exploratory testing (ET), 65, 71, 74
 exporting virtual machines, 378
 external user access, bug tracking system, 191

F

facilitating testing, team for, 379
 Fagan inspections, 380
 Fail Perfectly attribute (BVTs), 283
 Fail result (automated tests), 242
 failure analysis, automatic, 365–371
 failure count (test case metric), 217
 failure criteria in test cases, 213
 failure databases, 367
 failure matching, 367
 false alarms (metric), 357
 false negatives, with automated testing, 221
 false positives, 157, 247
 with automated testing, 221
 falsification tests, 69
 fan-in and fan-out measurements, 154
 fast rollbacks with services, 339
 feature area, bug, 193
 feature crews (Agile methodologies), 48
 features
 bugs vs., 197
 in milestone criteria, 47
 of services, 336

feedback systems. *See* customer feedback systems

Fiddler tool, 352, 354

field replaceable units (FRUs), 321

film industry, product development as, 11

finite state machines (FSMs), 161

finite state models, building, 166

fix if time (bug priority), 198

fix number method (WER), 308

fixed (bug resolution value), 195

fixed-constant values, 91

fixed-variable values, 91

font size (accessibility), 268

forgotten steps in test cases, 213

formal code reviews, 380

forums, Microsoft, 312

forward thinking in testing, 365–370

foundation (platform) services, 333

frequency of release, services, 336

frequency of testing

- performance testing, 253
 - as test case attribute, 212

Friday the 13th bug, 85

Frink, Lloyd, 21

frowny icon (Send a Smile), 310

FRUs (field replaceable units), 321

FSMs (finite state machines), 161

full automated service deployments, 337

fully automated tests. *See* automated tests

functional testing, 73–114

- boundary value analysis (BVA), 90–100
 - defining boundary values, 90
 - hidden boundaries, 97
 - using ECP tables with, 88, 93
- combinatorial analysis
 - effectiveness of, 111
 - tools for, in practice, 104–111
- equivalence class partitioning (ECP), 78–90
 - analyzing parameter subsets, 84–86
 - boundary condition tests, 88
 - boundary value analysis with, 88, 93
 - decomposing variable data, 80–82
 - example of, 83
 - need for, 74–78
 - vs. non-functional testing, 249
 - structural testing vs., 115
- functions, testing. *See* structural testing
- future of testing, 365–382
- fuzz testing, 271
- fuzzy matching, 221
- FxCop utility, 290

G

game data, collecting, 303

gatekeeper (check-in system), 287

Gates, Bill, 5, 11, 22, 319

gauntlet (check-in system), 287

General Manager of Test, 38

George, Grant, 25

getting to done (Agile methodologies), 49

gimmicks, test techniques as, 78

glass box testing, 71

global company, Microsoft as, 17

goals

- for milestones, 46, 49
- for performance testing, 253
- for usability testing, 270

grammar models, 170

graph theory, 166

gray box testing, 71

Group Test Managers, 38

grouping bugs in bug reports, 199

groups of variables in ECP, 82

H

Halo 2 game, 303

Halo 3 game, 8

Halstead metrics, 152–153

happy path, testing, 69

hard-coded paths in tests, 247

hardware

- accessible technology tools, 266
- device simulation framework, 234
- USB cart of death, 257

help phase (SEARCH test automation), 229, 244

helping testers, team for, 379

heuristics for equivalence class partitioning, 82

hidden boundary conditions, 97, 142

high-contrast mode, 268

hiring testers at Microsoft, 27

- campus recruiting, 29–31
- learning to be SDETs, 32

historical data, in test time estimation, 64

historical reference, test case as, 211

hotfixes, 155

how found (bug report field), 195

humorous bugs, 207–209

Hutcheson, Marnie, 117

Hyper-V, 375

I

Accessible interface, 226
 IC Testers, 35
 ICs (individual contributors), 33
 identification number validation, 139
ilities, list of, 250. *See also* non-functional testing
 imaging technology, 233
 impact, 5
 impact (competency), 28, 31
 impact of bug on customer, 194
 importing virtual machines, 378
 incubation, 11–14
 industry recruiting, 31
 influence (competency), 28, 31
 informal code reviews, 380
 initial build process, about, 282
 initiation phase (stress testing), 258
 innovation
 customer focus, 28. *See also* customer feedback systems
 incubation, 11–14
 in testing, 382
 innovation in PC computing, 329
 inputs (test cases), 212
 fuzz testing, 271
 Inspect Objects tool, 268
 installation testing, 233
 INT environment, 343–344, 345
 integrated services test environment, 343–344, 345
 integration testing (services), 346
 interactions within systems, 30
 International Project Engineering (IPE), 16
 internationalization, 17
 Internet memo, 319
 Internet Services as Microsoft focus, 319
 Internet Services Business Unit (ISBU), 12
 interoperability of bug tracking system, 191
 interpersonal awareness (competency), 29
 interpreting test case results, 217
 interviewing for tester positions, 29
 introduction (test strategy attribute), 66
 invalid class data (ECP), 81
 involvement with test automation, 220
 IPE (International Project Engineering), 16
 ISBU (Internet Services Business Unit), 12
 ISO 9000 program, 53
 issue type (bug report field), 195
 iterations, Agile methodologies, 50

J

jargon in test cases, 213
 JIT debuggers, 259
 job titles for software test engineers, 23
 moving from SDEs to SDETs, 24–27
 SDET IC, 35
 Test Architect, 34, 373–377
 in test management, 36
 Test Manager, 38
 Jorgensen's formula, 92
 Juran, Joseph, 366
 just-in-time (JIT) debuggers, 259

K

key scenario (test strategy attributes), 66
 keyboard accessibility. *See* accessibility testing
 keystrokes, simulating, 224
 K₂½nigsberg problem, 166
 knowledge testability, 67

L

large-scale test automation, 246
 Last Known Good (LKG) release, 55
 layered services, 327, 332
 Lead Software Development Engineering in
 Test title, 37
 leadership, 370
 Leads. *See* SDET Leads
 Lean program, 53
 learning how to be SDETs, 32
 legacy client bugs, 360
 legal defense, bug reports as, 192
 Length metric (Halstead), 152
 length of program (lines of code), 147–148
 libraries, 294
 libraries of applications for compatibility
 testing, 261, 385
 lifecycle, bugs, 189–190
 lifetime of automated tests, 220
 limitations of test patterns, 63
 line metrics (churn metrics), 273
 linearly independent basic paths, 133
 lines of code (LOC), 147–148
 Live Mail service, 8, 326
 Live Mesh, 320
 LKG (Last Known Good) release, 55
 load tests, 252, 256. *See also* performance testing; stress testing
 Office Online, 347

LOC (lines of code), 147–148
 Localization (IPE) discipline, 16
 log file parsers, 243, 370
 log files generated with test automation, 238, 243
 using for failure matching, 368
 long-haul tests, 252
 look-and-feel testing, 115, 116
 loop structures
 boundary testing of, 97, 99
 structural testing of, 128
 loosely coupled services, 333–335, 346
 love bug, 208
 low-resource testing, 256
 Luhn formula, 139

M

machine roles, 341
 machine virtualization, 372–379
 managing failures during tests, 377
 test scenarios, 374–377, 379
 maintainability testing, 250
 testability, 67–69
 maintainability of source code, complexity and, 156
 managed code analysis, 290
 managed code test attributes, 237
 management career paths, 33
 test management, 36
 managing bugs, 187–209
 attributes of tracking systems, 190
 bug bars (limits on bugs assigned), 205–209
 bug lifecycle, 189–190
 bug reports, 192–197
 common mistakes in, 198–201
 using data effectively, 201–205
 bug workflow, 188
 classic Microsoft bugs, 207–209
 false positives, 157, 247
 with automated testing, 221
 triage, 196–198
 managing test cases. *See* test cases
 mandatory practices, 56
 manual testing, 71, 213. *See also* exploratory testing
 mashups, 328, 349
 matching failures, 367
 MBD (Microsoft Business Division), 5
 MBT. *See* model-based testing
 McCabe, Thomas, 133, 149
 mean time between failure (MTBF) testing, 256
 measuring code complexity. *See* code complexity
 measuring performance, 253–255
 memory usage attribute (stress tests), 260
 message loops, 156
 metrics
 on bugs, 200
 as performance metrics, 204
 quota on finding, 205
 for code churn, 273
 code complexity
 Halstead metrics, 152–153
 how to use, 157
 object-oriented metrics, 153
 on defect detection. *See* DDE
 on emotional response, 310
 for performance, 253–255
 services, 351–356
 on quality, 300
 for quality of services (QoS), 357
 smoke alarm metrics, 155
 on test cases, 217
 Microsoft Active Accessibility (MSAA), 226
 Microsoft Application Verifier tool, 262
 Microsoft CIS (Cloud Infrastructure Services), 320
 Microsoft Connect, 312
 Microsoft Office, about, 8
 Microsoft Office Online, 334
 Microsoft OneNote customer connections, 314
 Microsoft Passport parental controlled, 334
 Microsoft Surface, 13
 Microsoft Test Leadership Team (MSTLT), 370, 382
 Microsoft Tester Center, 380
 Microsoft UI Automation framework, 226
 Microsoft Visual Studio 2008, Spec Explorer for, 175
 Microsoft Visual Studio Team Foundation Server (TFS), 264
 microsoft.public.* newsgroups, 312
 milestones, 45–48
 in Agile methodologies, 50
 quality milestone, 47
 milk carton bug, 207
 missing steps in test cases, 213
 mission statements, Microsoft, 4
 mistakes in test cases, 213–214

- mixed mode service upgrades, 340
- mod 10 checksum algorithm, 139
- model-based testing (MBT), 159–183
 - basics of modeling, 160
 - testing with model, 161–172
 - automating models, 166–171
 - designing models, 161
 - finite state models, 166
 - tips for modeling, 182
 - tools for, 174–182
- modeling control flow, 118, 122
- modeling threats, 271
- modeling *without* testing, 172, 182
- models for engineering workforce, 8
- models for software development, 42–45
- monitoring code changes. *See* source control
- monitoring code changes (churn), 273–275
- monitoring code review effectiveness, 381–384
- monitoring performance, 254
- monkey testing, 169
- mouse clicks, simulating, 224
- mouse target size (accessibility), 268
- movie industry, product development as, 11
- moving quality upstream, 366
- MQ (quality milestone), 47
 - services, 356
- MSAA (Microsoft Active Accessibility), 226
- MsaVerify tool, 268
- MSN (Microsoft Network), 320
- MSTLT (Microsoft Test Leadership Team), 370, 382
- MTBF testing, 256
- Muir, Marrin, 26
- multiclient stress tests, 260
- multiple bugs in single report, 199
- must fix (bug priority), 198
- Myers, Glenford, 82

N

- names for software test positions, 23
 - moving from SDEs to SDETs, 24–27
 - SDET IC, 35
 - Test Architect, 34, 373–377
 - in test management, 36
 - Test Manager, 38
- names for test patterns, 63
- naming service releases, 336
- native code analysis, 288
- negative testing, 108

- NEO (New Employee Orientation), 32
- Net Promoter score, 357
- network topology testing, 375
- new employee orientation (NEO), 32
- newsgroups, Microsoft, 312
- nodes (control flow graphs), 150
- no-known failure attribute (stress tests), 260
- non-functional (behavioral) testing, 116, 249–272
 - automating, 223
- non-functional testing
 - accessibility testing, 265–268
 - compatibility testing, 261–263
 - dogfooding (being users), 264, 350
 - dogfooding, 264, 350
 - performance testing, 231, 252–255
 - compatibility testing, 261–263
 - dogfooding (being users), 264, 350
 - how to measure performance, 253–255
 - Office Online, 347
 - in other testing situations, 254
 - services, metrics for, 351–356
 - stress testing, 257–260
 - security testing, 250, 270–272
 - stress testing, 257–260
 - architecture for, 258–260
 - Office Online newsgroups, 347
 - team organization, 251
 - usability testing, 250, 269
 - accessibility testing, 265–268
- not repro (bug resolution value), 195
- notification of bug assignment, 191
- number of passes or failures (test case metric), 217
- number of tests
 - boundary value analysis (BVA), 92
 - cyclomatic complexity and, 133
 - for performance testing, 253
 - reducing with data partitioning. *See* equivalence class partitioning (ECP)
- n*-wise testing, 102
 - effectiveness of, 111
 - in practice, 104–111
 - sufficiency of, 111

O

- object model, 224
- object-oriented metrics, 153
- observable testability, 67

offering tester positions to candidates, 29
 Office, about, 8
 Office Live. *See* Office Online
 Office Online, 334, 347
 Office Shared Services (OSS) team, 9
 OLSB (Online Live Small Business), 325
 one-box test platform (services), 340, 345
 OneNote customer connections, 314
 online services. *See* services
 open development, 386
 open office space, 381
 operating systems. *See* Windows operating systems
 Operations (Ops) discipline, 15
 operators in programs, counting. *See* Halstead metrics
 oracle (test pattern attribute), 63
 oracles, 240, 241
 organization of engineering workforce, 8
 orientation for new employees, 32
 orthogonal arrays (OA) approach, combinatorial testing, 102
 OSS (Office Shared Services) team, 9
 outdated test cases, 211
 output matrix randomization (PICT tool), 111
 overgeneralization of variable data, 80
 ownership of quality, 368
 Ozzie, Ray, 6, 319

P

packaged product vs. services, 325
 page load time metrics, 351, 352
 page weight (metric), 352, 353
 pair programming, 380
 pair testing, 72
 pair-wise analysis, 102
 effectiveness of, 111
 insufficiency of, 111
 in practice, 104–111
 parameter interaction testing. *See* combinatorial analysis
 parental controlled with WLID, 334
 Pareto, Vilfredo, 145
 Pareto principle, 145
 parsing automatic test logs, 243, 370
 partial production upgrades, services, 338
 partitioning data into classes. *See* decomposing variable data (ECP)
 Partner SDETs, 34

Partner Software Development Engineer in Test titles, 35
 pass count (test case metric), 217
 pass rate (test case metric), 217, 243
 Pass result (automated tests), 242
 pass/fail criteria in test cases, 213
 passion for quality (competency), 28
 path testing. *See* basis path testing
 patterns-based testing approach, 62–64
 PC computing innovations, waves of, 329
 PDCA cycle, 51
 percentage of false alarms (metric), 357
 percentage of tickets resolved (metric), 357
 perception of quality, 298
 perf and scale clusters, 342, 345
 Perfmon.exe utility, 254
 performance. *See also* metrics
 browser-based service performance metrics, 351–356
 bug data as metrics of, 204
 metrics for services, 351–356
 in milestone criteria, 47
 quality gates, 49
 services and processing power, 323
 performance counters, 254
 performance testing, 231, 252–255
 compatibility testing, 261–263
 dogfooding (being users), 264, 350
 how to measure performance, 253–255
 Office Online, 347
 in other testing situations, 254
 services, metrics for, 351–356
 stress testing, 257–260
 performing arts organization, Microsoft as, 10
 personas for accessibility testing, 266
 pesticide paradox, 77
 Petri nets, 173
 PICT tool, 104
 pitfalls with test patterns, 63
 Plan phase (PDCA cycle), 51
 planning, 50
 for performance bottlenecks, 254
 for services testing, 329
 for test automation, 232
 Platform Products and Services Division. *See* PSD
 platform services, 332
 platforms for test automation, 221
 play production, shipping products as, 10
 point of involvement with test automation, 220

portability testing, 250
 postbuild testing, 287
 postponed bugs, 189, 195
 power, growth and, 323
 practical baseline path technique, 134
 prebuild testing, 287
 predicted results (test case attribute), 212
 predicting quality perception, 298
 PRefast tool, 288
 Principle SDETs, 34
 Principle Software Development Engineer in
 Test titles, 35
 Principle Test Managers, 38
 Print Verifier, 263
 prioritizing bugs, 189, 196–198
 bug severity, 194
 Send a Smile program and, 311
 proactive approach to testing, 368
 problem (test pattern attribute), 63
 Problem Reports and Solutions panel, 305
 problem solving competency, 28
 process improvement, 51–53, 281
 formal systems for, 52
 services, 356
 processing power for services, 323
 product code. *See* entries at code
 product engineering disciplines, 15, 21
 product engineering divisions at Microsoft, 5
 product releases. *See* releases
 Product Studio, 187
 product support, 26
 product teams, 9
 Product Unit Manager (PUM) model, 8
 production, testing against, 349–351
 program decisions, counting. *See* cyclomatic
 complexity, measuring
 program length, measuring, 147–148
 Program Management (PM) discipline, 15, 21
 programmatic accessibility, 265, 268
 progress tracking, 211
 Project Atlas, 6
 project management
 bug prioritization, 198
 competency in, 28
 prototypes, 44
 PSD (Platform Products and Services Division),
 5
 PUM (Product Unit Manager) model, 8
 purpose (test case attribute), 212

Q

QA (quality assurance), 368
 QoS (quality of service) programs, 356
 quality, cost of, 369
 quality, passion for (competency), 28
 quality, service, 336
 quality assurance (QA), 368
 quality culture, 366
 quality gates, 49
 quality metrics, 300
 quality milestone, 47
 services, 356
 quality of service (QoS) programs, 356
 quality perception, 298
 quality tests. *See* non-functional testing
 Quests, 13
 quotas for finding bugs, 205

R

rack units (rack SKUs), 321
 Raikes, Jeff, 5
 random model-based testing, 169
 random selection, combinatorial testing, 101
 random walk traversals, 165
 random walk traversals (graph theory), 167
 ranges of values in ECP, 82
 RCA (root cause analysis), 357
 reactive approach to testing, 368
 reasons for code change, documenting, 278
 recruiting testers
 campus recruiting, 29–31
 industry recruiting, 31
 RedDog. *See* CIS (Cloud Infrastructure Services)
 Redmond workforce, about, 17
 regression tests, 220
 regular expressions, 170
 Rehabilitation Act Section 508, 265
 related test patterns, identifying, 63
 release/free builds, 55
 releases
 Microsoft-speak for, 55
 responsibility for managing, 56
 of services, frequency and naming of, 336
 reliability of bug tracking system, 191
 reliability testing, 250, 252
 milestone criteria, 46
 repeatability, test cases, 211
 repetition testing, 256
 reporting bugs. *See* managing bugs

reporting phase (SEARCH test automation)

reporting phase (SEARCH test automation), 229, 243
 reporting user data. *See* customer feedback systems
 reproduction steps (repro steps), 193
 Research discipline, 16
 resolution (in bug reports), 195
 resolved (bug status), 193
 resource utilization, 254
 low-resource and capacity testing, 256
 response from customers. *See* customer feedback systems
 responsiveness measurements, 253
 result types for automated testing, 242
 reusing code, 385–387
 reviewing automated test results, 240
 risk analysis modeling, 172
 risk estimation with churn metrics, 274
 risk management with services deployment, 338
 risk with code complexity, 145–158
 risk-based testing, 145
 role of testing, 370
 rolling builds, 285
 rolling upgrades, services, 339
 round trip analysis (metric), 352, 355
 Royce, Winston, 43
 Rudder, Eric, 33
 run infinitely attribute (stress tests), 260

S

S+S. *See* Software Plus Services (S+S)
 SaaS (software as a service), 326. *See also* Software Plus Services (S+S)
 sanitizing data before testing, 351
 scalability testing, 250, 252
 scale out (processing power), 343
 scale up (system data), 342
 scenario voting, 315
 scheduling, 50
 code reviews, 383
 debugging scope and, 278
 test automation, 221
 test case design, 70
 SCM. *See* source control
 scope of debugging, 277
 scope of testing, 70
 automated tests, 232
 scripted tests, code coverage of, 117

SDE. *See* Development (SDE) discipline; Test (SDET) discipline
 SDET Leads, 37
 SDET Managers, 38
 SDETs (Software Development Engineers in Test), 24–27
 learning how to be, 32
 recruiting. *See* recruiting testers
 using triangle simulations, 76
 SEARCH acronym for test automation, 229, 232–244
 analysis phase, 240–242
 cleanup phase, 243
 execution phase, 233–240
 help phase, 244
 reporting phase, 243
 setup phase, 232–234
 Section 508 (Rehabilitation Act), 265
 security
 data sanitization, 351
 quality gates, 49
 testing, 250, 270–272
 self-host builds, 55
 self-test build, 55
 self-toast builds, 55
 semiautomated tests, 213
 Send a Smile program, 310
 Senior SDET Leads, 38
 Senior SDET Manager, 38
 Senior SDETs, 34
 Senior Software Development Engineer in Test titles, 35
 servers, building services on, 331
 service groups, 321
 services, 317–362
 dogfooding, 350
 loose vs. tight coupling, 333–335, 346
 Microsoft services strategy, 318
 packaged product vs., 325
 performance test metrics, 351–356
 platform vs. top-level, 332
 processing power requirements, 323
 S+S testing approaches, 329–337
 S+S testing techniques, 337–356
 deployment automation, 337–339
 performance test metrics, 351–356
 test environment, 339–345
 testing against production, 349–351
 stand-alone and layered services, 327
 stateless vs. stateful, 335

- services (*continued*)
 - testing S+S
 - approaches for, 330–337
 - common bugs, 360
 - continuous quality improvement, 356–360
 - Services memo, 319
 - setup phase (SEARCH test automation), 229, 232–234
 - Seven Bridges of Königsberg problem, 166
 - severity, bug, 194
 - shared libraries, 294, 378
 - Shared Team model, 9
 - shared teams, 9
 - shared test clusters, 342, 344
 - sharing test tools, 294, 378, 386
 - Shewhart cycle, 51
 - ship room, shipping software from, 54–56
 - shipping software, 54–56
 - shirts, ordering new, 6
 - shortest path traversal (graph theory), 167
 - should fix (bug priority), 198
 - shrink-wrap software, 325
 - simple testability, 67
 - simplicity. *See* code complexity
 - simplified baseline path technique, 134
 - simplified control flow diagrams (CFDs), 122
 - single fault assumption, 89
 - Six Sigma program, 53
 - size issues (accessibility), 268
 - size of Microsoft engineering workforce, 27
 - campus recruiting, 29–31
 - Skip result (automated tests), 242, 243
 - SMEs as testers, 23
 - smiley icon (Send a Smile), 310
 - Smith, Brad, 6
 - smoke alarm metrics, 155
 - smoke tests, 283, 349
 - snapshots of code, 275, 374, 378
 - SOCK mnemonic for testability, 67
 - software as services. *See* services
 - software design, importance of, 61
 - Software Development Engineer in Test
 - Manager title, 37
 - Software Development Engineer in Test titles, 35
 - software engineering at Microsoft, 41–50
 - Agile methodologies, 44, 48–50
 - milestones, 45–48
 - traditional models, 42–45
 - software features
 - bugs vs., 197
 - in milestone criteria, 47
 - software libraries, 385
 - Software Plus Services (S+S), 318, 329–337. *See also* services
 - common bugs with, 360
 - continuous quality improvement, 356–360
 - vs. SaaS (software as a service), 326
 - testing approaches, 330–337
 - client support, 331
 - loose vs. tight coupling, 333–335, 346
 - platform vs. top-level, 332
 - release frequency and naming, 336
 - server builds, 331
 - stateless vs. stateful, 335
 - time-to-market considerations, 336
 - testing techniques, 337–356
 - deployment automation, 337–339
 - performance test metrics, 351–356
 - test environment, 339–345
 - testing against production, 349–351
 - software reliability. *See* reliability testing
 - software test engineers, 21–39. *See also* titles
 - for software test engineers
 - career paths in Test, 34–38
 - engineering careers, 33
 - learning how to be, 32
 - recruiting. *See* recruiting testers
 - sound features, accessibility of, 268
 - source (bug report field), 195
 - source control, 275–281
 - breaking builds, 285
 - check-in systems, 286
 - reasons for code changes, 278
 - Spec Explorer tool, 174–178
 - Windows 7 and, 181
 - special values in ECP, 82
 - specifications for test design, 66, 68
 - spiral model, 43
 - SQEs (software quality engineers), 25
 - stand-alone applications, 234
 - stand-alone services, 327
 - stapler stress, 257
 - starting state (model), 160
 - starting the test process, 65
 - state-based models. *See* model-based testing (MBT)
 - stateless vs. stateful services, 335

statement testing, 118
 static analysis, 56, 288–294
 managed code analysis, 290
 native code analysis, 288
 test code analysis, 292
 status, bug, 193
 Step attribute, 237
 steps in test cases, 212, 213
 STEs (Software Test Engineers), 24
 Stobie, Keith, 29, 36, 229
 stopwatch testing, 252
 strategic insight (competence), 28
 strategy, test, 66
 stress testing, 257–260
 architecture for, 258–260
 Office Online newsgroups, 347
 structural testing, 115–143
 basis path testing, 117–142
 block testing, 118–126
 condition testing, 129–132
 decision testing, 126–129
 functional testing vs., 115
 need for, 116
 subject matter experts as testers, 23
 support, product, 26
 support for test automation, 221
 SupportFile attribute, 237
 Surface, 13
 switch/case statement, testing, 122
 syntax elements in programs, counting. *See*
 Halstead metrics
 syntax errors, 284
 systematic evaluation approaches,
 combinatorial testing, 101
 systems, test. *See* test tools and systems
 systems interactions, 30
 system-wide accessibility settings, 267

T

TAG (Test Architect Group), 373–377, 382
 TAs. *See* Test Architects
 TCMs (test case managers), 209, 215, 217
 cross-referencing with automation scripts,
 246
 TDSs (test design specifications), 66, 68
 Team Foundation Server (TFS), 264
 teams
 feature crews (Agile methodologies), 48
 for non-functional testing, 251
 open office space, 381
 usability labs, 269
 war team, 54
 Windows Stress team, 260
 technical excellence (competency), 28
 Technical Fellows, 34
 techniques as gimmicks, 78
 templates for sharing test patterns, 63
 Test (SDET) discipline, 15. *See also* SDETs
 (Software Development Engineers in Test)
 Test A Little attribute (BVTs), 283
 test architects, 382
 Test Architects (TAs), 34, 373–377
 test automation, 219–248
 automatic failure analysis, 365–371
 developing, 30
 elements of testing, 228–231
 exploratory testing vs., 71
 running the tests, 245–247
 common mistakes with, 247
 large-scale testing, 246
 SEARCH acronym for test automation, 229,
 232–244
 analysis phase, 240–242
 cleanup phase, 243
 execution phase, 233–240
 help phase, 244
 reporting phase, 243
 setup phase, 232–234
 test code analysis, 292
 testing, 61
 tracking changes in, 275
 user interface automation, 223–228
 value of, 219–223
 Test Broadly Not Deeply attribute (BVTs), 283
 test case managers (TCMs), 209, 215, 217
 cross-referencing with automation scripts,
 246
 test cases, 209–217, 215, 217
 analysis of, 293
 anatomy of, 212
 common mistakes with, 213–214
 counting, 215–216
 cross-referencing with automation scripts,
 246
 defined, 209, 216
 design of, 61–72
 best practices, 61
 estimating test time, 64
 getting started, 65–67

- test cases (*continued*)
 - practical considerations, 70–72
 - testability, 67–69
 - testing good and bad, 69
 - using test patterns, 62–64
- documenting with automated testing, 244
- executing automatically, 234, 246. *See also*
 - test automation
- milestone criteria, 46
- tracking and interpreting results (metrics), 217
- tracking changes in, 275
- value of, 211
- workflow, 188
- test cleanup, 243
- test clusters, 341, 345
 - deployment test clusters, 344, 345
 - dogfood clusters, 350
 - shared, 342, 344
- test code analysis, 292
- test code snapshots, 275, 374, 378
- test collateral, 246
- test controllers, 246
- test coverage. *See* code coverage
- test data, creating with grammar models, 170
- test deliverables (test strategy attributes), 67
- test design, importance of, 61
- test design specifications (TDSs), 66, 68
- test environment for services, 339
- test excellence, 378–382
- Test Fast attribute (BVTs), 283
- test flags, 346
- test frequency
 - performance testing, 253
 - as test case attribute, 212
- test harnesses, 235, 244
- test innovation, 382
- test leadership, 370–376
- test logs (automated testing), 238, 243
 - using for failure matching, 368
- test matrix for services testing, 329
- test oracles, 240, 241
- test pass, defined, 216, 217
- test patterns, 62–64
- test points, 215, 216
- test run, defined, 216
- test strategies, 66
- test suite, defined, 216
- test therapists, 380
- test time, estimating, 64
- test tools and systems
 - automation. *See* test automation
 - bug management. *See* managing bugs
 - build process, 281–287
 - breaking builds, 284
 - testing daily builds, 374
 - code churn, 273–275
 - customer feedback systems, 297–315
 - connecting with customers, 312–315
 - emotional response, 309–312
 - for services, 357
 - testing vs. quality, 297–299
 - watching customers. *See* CEIP (Customer Experience Improvement Plan)
 - Windows Error Reporting, 304–309
 - miscellaneous, 294
 - non-functional testing. *See* non-functional testing
 - source control, 275–281
 - breaking builds, 285
 - check-in systems, 286
 - reasons for code changes, 278
 - static analysis, 56, 288–294
 - managed code analysis, 290
 - native code analysis, 288
 - test code analysis, 292
- testability, 67–69. *See also* maintainability
 - testing
- TestCleanup attribute, 237
- Tester Center, 380
- tester DNA, 23, 25
- testing, future of, 365–382
- testing against production, 349–351
- Testing at Microsoft for SDETs class, 32
- testing coverage
 - analysis tools for, 293
 - behavioral and exploratory testing, 117
 - combinatorial analysis and, 112
 - with functional testing, 77
 - milestone criteria, 46
 - quality gates, 49
 - scripted tests, 117
 - statement vs. block coverage, 118
- testing techniques as gimmicks, 78
- testing the tests, 288, 292
- testing with models. *See* model-based testing (MBT)
- TestInitialize attribute, 237
- TestMethod attribute, 237
- text matrix for automated testing, 232
- TFS (Team Foundation Server), 264

ThinkWeek, 13
 threat modeling, 271
 3(BC) formula, 94, 100
 threshold method (WER), 308
 tickets resolved (metric), 357
 tidal wave bug, 361
 tightly coupled services, 333–335, 346
 time for code reviews, monitoring, 383
 time investment. *See* scheduling
 time to detection (metric), 357
 time to document, 211
 time to market, services, 336
 time to resolution (metric), 357
 titles for software test engineers, 23
 moving from SDEs to SDETs, 24–27
 SDET IC, 35
 Test Architect, 34, 373–377
 in test management, 36
 Test Manager, 38
 titles of bugs (in bug reports), 192
 tool sharing, 294, 378, 386
 tools for accessibility technology, 266, 268
 top-level services, 332
 topology testing, 375
 tracking bugs. *See* managing bugs
 tracking code changes. *See* source control
 tracking code review data, 384
 tracking test cases, 217
 tracking test progress, 211
 training as SDETs, 32
 training strategy in test design, 66, 67
 transitions (in models), 160
 Petri nets, 173
 trends in test failures, analyzing, 371
 triad (Test, Development, Program Management), 16
 triage for managing bugs, 189, 196–198
 bug severity, 194
 Send a Smile program and, 311
 triangle simulation (Weinberg's triangle), 76
 Trudau, Garry, 6
 Trustworthy attribute (BVTs), 283
 t-shirts, ordering new, 6
 tsunami effect, 361
 Turner, Kevin, 6

U

UI automation, 223–228
 brute force approach, 227
 uncertainty, reducing with BGM, 172
 unique tools for unique problems, 294
 uniqueness of values in ECP, 82
 unit tests, block testing for, 122
 universities, recruiting from, 29
 upgrading services, 338
 Usability and Design discipline, 15, 21
 usability testing, 250, 269
 accessibility testing, 265–268
 usage data, collecting. *See* CEIP (Customer Experience Improvement Plan)
 USB cart of death, 257
 User Assistance and Education. *See* Content discipline
 user interface
 automation of, 223–228
 programmatic access to, 265, 268
 users, being. *See* dogfooding
 UX. *See* Usability and Design discipline

V

valid class data (ECP), 81
 values, Microsoft, 4
 variable data, decomposing (ECP), 80–82
 venture capital teams, internal, 12
 verbose, test cases as, 213
 verification tests, 69
 version number, bugs, 193
 Vice President of Test, 38
 VINCE (Verification of Initial Consumer Experience), 303
 Virtual Earth, 8
 virtual teams for non-functional testing, 251
 virtualization, 372–379
 managing failures during tests, 377
 test scenarios, 374–377, 379
 visual freeze, defined, 55
 Visual Round Trip Analyzer (VRTA), 352, 355
 Visual Studio 2008, Spec Explorer for, 175
 Visual Studio Team Foundation Server (TFS), 264
 VMs. *See* machine virtualization

voice of customer, 357
Voodoo Vince, 303
VPNs (virtual private networks), 68

W

war room, shipping software from, 54–56
Warn result (automated tests), 242
watching customers. *See* CEIP (Customer Experience Improvement Plan)
waterfall model, 42
waves of innovation in PC computing, 329
Web services. *See* services
weighted methods per class (metric), 153
weighted traversals (graph theory), 167
Weinberg's triangle, 76
WER (Windows Error Reporting), 304–309
White, David, 26
white box testing, 71
 assumption of bias in (false), 116
 structure testing as, 115
Whittaker, James, 36
Whitten, Greg, 22
Windows 7, Spec Explorer and, 181
Windows 95, supporting, 27

Windows Error Reporting (WER), 304–309
Windows Live ID (WLID), 334
Windows Live Mail service, 8, 326
Windows Mobile, about, 8
Windows operating systems
 about, 8
 accessibility settings, 265
Windows Powered Smart Display, 227
Windows Sustained Engineering (SE) Team, 155
WMC metric, 153
“won't fix” bugs, 69, 147, 189
workflow for bugs, 188
workforce size at Microsoft, 27
 campus recruiting, 29–31

X

Xbox 360, about, 8

Z

zero bug bounce, 198
zero bugs concept, 198

