LEAN-AGILE ACCEPTANCE TEST-DRIVEN DEVELOPMENT

Better Software Through Collaboration

Net Objectives

Lean-Agile Series

KEN PUGH

Praise for *Lean-Agile Acceptance Test-Driven Development*

"*Lean-Agile Acceptance Test-Driven Development* tells a tale about three fictive project stakeholders as they use agile techniques to plan and execute their project. The format works well for the book; this book is easy to read, easy to understand, and easy to apply."

—*Johannes Brodwall, Chief Scientist, Steria Norway*

"Agile development, some say, is all about pairing, and, yes, I'm a believer in the power of pairing. After reading this book, however, I became a fan of the 'triad'—the customer or business analyst + the developer + the tester, who work collaboratively on acceptance tests to drive software development. I've written some patterns for customer interaction and some patterns for testing and I like what Ken Pugh has chosen to share with his readers in this down-to-earth, easy-to-read book. It's a book full of stories, real case studies, and his own good experience. Wisdom worth reading!"

—*Linda Rising, Coauthor of* Fearless Change: Patterns for Introducing New Ideas

"The Agile Manifesto, Extreme Programming, User Stories, and Test-Driven Development have enabled tremendous gains in software development; however, they're not enough. The question now becomes 'How can I ensure clear requirements, correct implementation, complete test coverage, and more importantly, customer satisfaction and acceptance?' The missing link is acceptance as defined by the customer in their own domain language. *Lean-Agile Acceptance Test-Driven Development* is the answer."

—*Bob Bogetti, Lead Systems Designer, Baxter Healthcare*

"Ken Pugh's *Lean-Agile Acceptance Test-Driven Development* shows you how to integrate essential requirements thinking, user acceptance tests and sounds, and lean-agile practices, so you can deliver product requirements correctly and efficiently. Ken's book shows you how table-driven specification, intertwined with requirements modeling, drives out acceptance criteria. *Lean-Agile Acceptance Test-Driven Development* is an essential guide for lean-agile team members to define clear, unambiguous requirements while also validating needs with acceptance tests."

—*Ellen Gottesdiener, EBG Consulting, www.ebgconsulting.com, Author of* Requirements by Collaboration *and* The Software Requirements Memory Jogger

“If you are serious about giving Agile Testing a chance and only have time to read one book, read this one.”

—*David Vydra, http://testdriven.com*

“This book provides clear, straightforward guidance on how to use business-facing tests to drive software development. I’m excited about the excellent information in this book. It’s a great combination of the author’s experiences, references to other experts and research, and an example project that covers many angles of ATDD. A wide range of readers will learn a lot that they can put to use, whether they work on projects that call themselves lean or agile or simply want to deliver the best possible software product.”

—*Lisa Crispin, Agile Tester, ePlan Services, Inc., Author of* Agile Testing

Lean-Agile Acceptance Test-Driven Development

This page intentionally left blank

Lean-Agile Acceptance Test-Driven Development

Better Software Through Collaboration

Ken Pugh

Addison-Wesley

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

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

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

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

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

For sales outside the United States, please contact:

International Sales
international@pearson.com

Visit us on the Web: informit.com/aw

Library of Congress Cataloging-in-Publication Data

Pugh, Kenneth.
Lean-agile acceptance test driven development : better software through collaboration / Ken Pugh.
p. cm.
Includes bibliographical references and index.
ISBN-13: 978-0-321-71408-4 (pbk. : alk. paper)
ISBN-10: 0-321-71408-3 (pbk. : alk. paper) 1. Agile software development.
2. Computer software--Testing. 3. Computer software—Quality control.
4. Cooperation. I. Title.
QA76.76.D47P837 2011
005.1'4--dc22
2010042906

Copyright © 2011 Pearson Education, Inc.

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

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

ISBN-13: 978-0-321-71408-4
ISBN-10: 0-321-71408-3

Text printed in the United States on recycled paper at R.R. Donnelley in Crawfordsville, Indiana.
First printing December 2010

Editor-in-Chief
Karen Gettman

Executive Editor
Chris Guzikowski

Senior Development Editor
Chris Zahn

Managing Editor
Kristy Hart

Project Editor
Jovana San Nicolas-Shirley

Copy Editor
Karen Gill

Indexer
Cheryl Lenser

Proofreader
Sheri Cain

Editorial Assistant
Raina Chrobak

Cover Designer
Alan Clements

Compositor
Nonie Ratcliff

*I'd like to dedicate this book to three people.
My brother Bob inspired me to become an engineer.
I recall one time when he was home from college and presented me
with the N-body problem [Wiki01] and the four color map problem
[Wiki02]. My high school science teacher, Mr. Sanderson, spurred
me on to explore topics such as why there is air. My mechanical
engineering professor at Duke, Dr. George Pearsall, encouraged
exploration. In his strength of materials class, I discovered
why my guitar strings broke. To each of them, I give thanks.*

This page intentionally left blank

Contents

Introduction 1

Part I: The Tale

Chapter 1: Prologue 9
- Ways to Develop Software 9
  - One Way 9
  - Another Way 9
  - The Difference 10
- The Importance of Acceptance Tests 10
- System and Team Introduction 12
  - The System 12
  - The People 13
- Summary 14

Chapter 2: Lean and Agile 15
- The Triad and Its Units 15
- Post-Implementation Tests 17
- Quick Feedback Better Than Slow Feedback 18
- Preimplementation Tests 19
- Lean and Agile Principles 20
- Summary 21

Chapter 3: Testing Strategy 23
- Types of Tests 23
- Where Tests Run 25
- Test Facets 26
  - Control and Observation Points 27
  - New Test Is a New Requirement 27
- Summary 28

Chapter 4: An Introductory Acceptance Test 29
A Sample Business Rule 29
Implementing the Acceptance Tests 31
Test Script 32
Test User Interface 33
xUnit Test 34
Automated Acceptance Test 35
An Overall Test 36
Testing Process 37
Summary 37

Chapter 5: The Example Project 39
The Charter 39
Objectives 40
Project Acceptance Tests 41
High-Level Requirements 43
Features 43
Feature Acceptance Criteria 45
Summary 46

Chapter 6: The User Story Technique 47
Stories 47
Features into Stories 48
Roles 49
Role Attributes 49
Persona 50
Stories for Roles 51
Story Acceptance Criteria 52
Acceptance Tests Determine Size 53
Customer Terms 54
INVEST Criteria 55
Summary 56

Chapter 7: Collaborating on Scenarios 57
Use Cases from User Stories 57
Simple Use Case 59
Exceptions and Alternatives 60
Acceptance Tests 63
Documentation 63

Story Map . . . 63
Conceptual Flow . . . 65
Communication . . . 66
Summary . . . 68

Chapter 8: Test Anatomy . . . 69
Triad Creates Tests . . . 69
Test Context . . . 70
Test Structure . . . 71
Calculation Table . . . 73
Data Table . . . 74
Action Table . . . 75
Tests with Example Values . . . 76
Requirements Revised . . . 77
Acceptance Test Revised . . . 78
Test with Values in Text . . . 79
When and Where Tests Are Run . . . 80
Summary . . . 81

Chapter 9: Scenario Tests . . . 83
Tests for Exception Scenarios . . . 83
Tests for Business Rules . . . 87
Cross-Story Issues . . . 88
Don't Automate Everything . . . 89
Multi-Level Tests . . . 90
User Interface Tests . . . 93
Check the Objectives . . . 93
Summary . . . 94

Chapter 10: User Story Breakup . . . 95
Acceptance Tests Help Break Up Stories . . . 95
Business Rule Tests . . . 96
A Story with a Business Rule . . . 100
Summary . . . 101

Chapter 11: System Boundary . . . 103
External Interfaces . . . 103
More Details . . . 107

External Interface Tests 108
Component Tests 108
Test Doubles and Mocks. 111
What Is Real?. 112
Story Map of Activities 113
Summary 114

Chapter 12: Development Review 115
The Rest of the Story 115
Usability Testing 116
Separating State from Display 116
Quality Attribute Tests 118
Workflow Tests. 119
Deployment Plans 120
From Charter to Deliverable 120
Summary 121

Part II: Details

Chapter 13: Simplification by Separation 125
Complex Business Rules. 125
Simplify by Separating. 126
The Simplified Rule. 128
Rental History 128
Summary 130

Chapter 14: Separate View from Model. 131
Decouple the User Interface 131
Decoupling Simplifies Testing 136
Summary 136

Chapter 15: Events, Responses, and States. 137
Events and an Event Table. 137
States and State Transitions 139
Internal State or External Response 142
Transient or Persistent States. 144
A Zen Question. 144
Summary 144

Chapter 16: Developer Acceptance Tests **145**
Component Acceptance Tests 145
Field Display Tests 145
Tabular Display Tests 147
Summary 151

Chapter 17: Decouple with Interfaces **153**
Tests for a Service Provider 153
The Interface 153
Quality Attribute Tests 155
Comparing Implementations 155
Separating User Interface from Service 157
Separation of Concerns 158
Reusable Business Rules 158
Summary 159

Chapter 18: Entities and Relationships **161**
Relationships 161
Entities and Relationships 161
Multiple Relationships 163
Alternative Representations 166
Summary 166

Chapter 19: Triads for Large Systems **167**
Large Systems 167
When a Customer Test May Not Be Required 169
Data Conversion 170
Database Conversions 170
What If There Are No Tests? 170
Legacy Systems 172
Summary 173

Part III: General Issues

Chapter 20: Business Capabilities, Rules, and Value **177**
Business Capabilities 177
Scenario Handling 178
Business Rules Exposed 179

A Different Business Value 179
Summary 181

Chapter 21: Test Presentation 183
Customer Understood Tables 183
Table Versus Text 185
Specifying Multiple Actions 185
Complex Data 187
Custom Table Forms 188
Summary 189

Chapter 22: Test Evaluation 191
Test Facets 191
Understandable to Customers 191
Spell Checked 192
Idempotent 192
Not Fragile 192
Test Sequence 193
Workflow Tests 193
Test Conditions 194
Separation of Concerns 194
Test Failure 195
Test Redundancy 196
No Implementation Issues 197
Points to Remember 197
Summary 198

Chapter 23: Using Tests for Other Things 199
Uses of Acceptance Tests 199
Degree of Doneness 199
Estimation Aid 200
Breaking Down Stories 200
Developer Stories 200
Tests as a Bug Report 201
Root Cause Analysis 201
Production Bugs 202
Regression Testing 202
Summary 202

Chapter 24: Context and Domain Language **205**
Ubiquitous Language 205
Two Domains 207
Summary 208

Chapter 25: Retrospective and Perspective **209**
Recap 209
The Process 210
Testing Layers 210
The Tests 211
Communication 212
What's the Block? 212
Monad 212
Unavailable Customer 213
Change 213
Risks 214
Benefits 214
Summary 215

Part IV Case Studies

Chapter 26: Case Study: Retirement Contributions **219**
Context 219
The Main Course Test 220
Setup 220
Event 221
Expected 221
Implementation Issues 222
Separation of Concerns 222
Business Value Tracking 223
One Exception 223
Event 223
Expected 224
Another Exception 225
Event 225
Expected 225

Two Simultaneous Exceptions . . . 226
Event . . . 226
Expected . . . 227
The Big Picture . . . 227
Event Table . . . 228
State Transition Table . . . 228
Summary . . . 230

Chapter 27: Case Study: Signal Processing . . . 231
It's Too Loud . . . 231
Sound Levels . . . 231
Developer Tests . . . 233
Summary . . . 233

Chapter 28: Case Study: A Library Print Server . . . 235
The Context . . . 235
A Workflow Test . . . 236
Summary . . . 241

Chapter 29: Case Study: Highly Available Platform . . . 243
Context for Switching Servers . . . 243
Test for Switching Servers . . . 244
Test for Technical Rule . . . 246
Summary . . . 248

Part V: Technical Topics

Chapter 30: How Does What You Do Fit with ATDD? . . . 251
Test Platforms . . . 251
Internal Design from Tests . . . 252
Device Testing . . . 254
Starting with User Interfaces . . . 255
Black Box Testing . . . 255
Unit Testing . . . 256
Summary . . . 256

Chapter 31: Test Setup . . . 257
A Common Setup . . . 257
Some Amelioration . . . 259

Test Order 260
Persistent Storage Issues 260
Summary 261

Chapter 32: Case Study: E-Mail Addresses 263
Context 263
Breaking Down Tests 264
Local-Part Validation 265
Domain Tests 266
Disallowed Domain Tests 268
Test to Ensure Connection 269
Verification Test 269
Summary 270

Part VI: Appendices

Appendix A: Other Issues 273
Context 273
Customer Examples 274
Fuzzy Acceptance Tests 274
Acceptance Test Detail 275
Requirements and Acceptance Tests 275
Documenting Requirements and Tests 276
Decoupling Requirements 276
Separation of Issues 276
Testing Systems with Random Events 277
The Power of Three 277
Summary 278

Appendix B: Estimating Business Value 279
Business Value 279
Developer Stories 281
Summary 282

Appendix C: Test Framework Examples 283
The Examples 283
Fit Implementation 284
Setup 284
Check-Out CD 284

Check-In 286
Category-Based Rental Fees 287
Slim—Table Style 288
Header 288
Setup 288
Check-Out CD 288
Check-In 290
Category-Based Rental Fees 291
Slim—Cucumber Style 291
Setup 291
Check-Out CD 292
Check-In CD 292
Scenario Library 292
Category-Based Rental Fees 294
Robot 295
Setup 295
Check-Out CD 295
Check-In CD 296
Category-Based Rental Fees 296
Cucumber 296
Check-Out CD 297
Check-In CD 297
Category-Based Rental Fees 297
Test Frameworks 298
Summary 298

Appendix D: Tables Everywhere 299

User Interface Tests with Tables 299
Requirement Tables 301
Another Table 302
Quality Attribute Requirements 303
Data Tables 304
Summary 304

Appendix E: Money with ATDD . . . 305
The Context . . . 305
The Original Tests . . . 306
The Acceptance Test Approach . . . 307
Summary . . . 310

Appendix F: Exercises . . . 311
Calculator . . . 311
Create Some Tests . . . 313
More Exercises . . . 313
Sam's CD Rental . . . 314
Triangle . . . 314
File Copying Exercise . . . 314

References . . . 315

Epilogue . . . 323

Index . . . 333

Acknowledgments

Over my two-fifths of a century in software, I've have the opportunity to interact with a wide range of people. Many of the ideas expressed in this book have come from them—from their books, their talks, and personal conversations. Albert Einstein said, "Creativity is knowing how to hide your sources." I would like not to hide these people. The only problem is I can't always remember what I got from whom. The list includes in no particular order: Cem Kaner, Jerry Weinberg, James Bach, Michael Bolton, Brian Marick, Ellen Gottesdiener, Karl Wiegers, Ward Cunningham, Jim Shore, Rick Mugridge, Lisa Crispin, Janet Gregory, Kent Beck, Gerard Meszaros, Alistair Cockburn, Andy Hunt, Bob Martin, Dale Emery, III, Michael Feathers, Mike Cohn, Jim Highsmith, Linda Rising, Ron Jeffries, Mary Poppendieck, Jim Coplien, Norm Kerth, Scott Ambler, Jared Richardson, Dave Thomas, Martin Fowler, Bill Wake, Tim Lister, Eric Evans, Bret Pettichord, Brian Lawrence, Jeff Patton, David Hussman, Rebecca Wirfs-Brock, Joshua Kerievsky, Laurie Williams, Don Gause, James Grenning, Tom DeMarco, Danny Faught, Jeff Sutherland, David Astels, Lee Copeland, Elisabeth Hendrickson, Bob Galen, Gary Evans, George Dinwiddie, Jutta Eckstein, Bob Hartman, David Chelimsky, Dan North, Lasse Koskela, Cedric Beust, and Larry Constantine.

I'd like to thank Rob Walsh of EnvisionWare for the case study of a library print server, Robert Martin for the Cucumber style example in Slim, Markus Gaertner for the Slim example, Dale Emery for the Robot example, and John Goodsen for the Cucumber example. I appreciate Gerard Meszaros for permission to use his testing matrix graphic. Thanks to Dawn Cannan, Gabriel Le Van, Stephen Cresswell, Jared Richardson, Ian Cooper, Greg McNelly, and Gary Marcos for their ATDD stories in the Epilogue. I'd like to acknowledge the Net Objectives gang: Alan Shalloway, Jim Trott, Scott Bain, Amir Kolsky, Cory Foy, and Alan Chedalawada. Also thanks to Omie and Tammi for keeping me sane.

In helping make this book a reality, I thank the people at Addison-Wesley, Pearson Technology Group: Chris Guzikowski, Chris Zahn, Raina Chrobak, Kristy Hart, Jovana San Nicolas-Shirley, Karen Gill, Nonie Ratcliff, Cheryl Lenser, and Sheri Cain. And to reviewers Andrew Binstock, Graham Oakes, Lisa Crispin, Linda Rising, Bill Wake, Robert Bogetti, Johannes Brodwall, Peter Kurpis, SGuy Ge, Tom Wessel, Kody Shepler, Jinny Batterson, Julian Harty, and III.

Last but not least, I thank Leslie Killeen, my wife. She is a weaver. Software is not her field. She reviewed my drafts, gave helpful hints, and supported me through the creation process.

About the Author

KATAHDIN
BAXTER PEAK ELEVATION – 5267 FT
NORTHERN TERMINUS OF THE
APPALACHIAN TRAIL
A MOUNTAIN FOOTPATH EXTENDING OVER
2000 MILES TO SPRINGER MTN. GEORGIA
THOREAU SPRING 1.0M
KATAHDIN STREAM CAMPGROUND 5.2
PENOBSCOT WEST BRANCH AT ABOL BRIDGE 14.5
MAINE-NEW HAMPSHIRE STATE LINE 281.4
MT. WASHINGTON, N. H. 332.4
SPRINGER MTN., GEORGIA 2100.2
MAINE APPALACHIAN TRAIL CLUB

Kenneth Pugh has over two-fifths of a century of software experience. Previously a principal at Pugh-Killeen Associates, he is now a fellow consultant for Net Objectives. He has developed software applications ranging from radar tracking to financial analysis. Responsibilities have included everything from gathering requirements to testing. After the start of the new millennium, he has worked with teams to create software more effectively with lean and agile processes. He has spoken at numerous national conferences; consulted and taught all over the world; and testified on technology topics. This is his seventh book. In 2006, his book *Prefactoring* won the Jolt Award [DrDobbs01]. In his spare time, he snowboards, windsurfs, and backpacks. Between 1997 and 2003, he completed the Appalachian Trail. The cover photograph of Mount Katahdin, the northern end of the trail, was taken by the author from Abol Bridge in Maine.

Introduction

"Context is all."

Margaret Atwood, *The Handmaid's Tale*

The context for the tale is introduced. A brief background of acceptance test–driven development (ATDD) is presented.

Testable Requirements

Developing software with testable requirements is the theme of this book. A testable requirement is one with an acceptance test. Acceptance tests drive the development of the software. As many development groups have experienced, creating acceptance tests prior to implementing requirements decreases defects and improves productivity. (See the Epilogue for examples.) A triad—the customer/business analyst, developer, and tester—collaborates on producing these tests to clarify what is to be done. In creating a high-quality product, ATDD is as much about this clarification as it is about the actual testing.

As an example, do you have criteria in mind as to whether this book will meet your needs? If you finish this book, how will you know whether it has met those criteria? This book represents an implementation of something that should meet your needs. Because you are reading this book after its completion, you don't have an opportunity to influence the acceptance criteria. But let me list the criteria here and see if this is what you are after.

In English classes, the teacher emphasized that a story should contain a who, what, when, where, why, and how. So I've made that the goal of this book. It explains

- Who creates acceptance tests
- What acceptance tests are
- When the acceptance tests should be created
- Where the acceptance tests are used
- Why acceptance test-driven development is beneficial
- How the acceptance tests are created

By the end of this book, the expectation is that you should understand how testable requirements can make the software development process more enjoyable (or at least less painful) and help in producing higher-quality products., Let's begin with a brief discussion on the why, what, where, and who issues.

Why ATDD Is Beneficial

Let's start with the answer to the why question. Jeff Sutherland, the cocreator of Scrum, has metrics on software productivity [Sutherland01]. He has found that adding a quality assurance person to the team and creating acceptance tests prior to implementation doubles the team's productivity. Your actual results may vary, but teams adopting ATDD have experienced productivity and quality increases. Mary Poppendieck says that creating tests before writing code is one of the two most effective and efficient process changes for producing quality code. (The other is frequent feedback.) [Poppendieck01] Customer-developer-tester collaboration reduces unnecessary loops in the development process. As Jerry Weinberg and Don Gause wrote, "Surprising, to some people, one of the most effective ways of testing requirements is with test cases very much like those for testing the completed system" [Weinberg01].

If you are going to test something and document those tests, it costs no more to document the tests up front than it does to document them at the end. But these are more than just tests. As stated in Chapter 3, "Testing Strategy," "The tests clarify and amplify the requirements." An acceptance test is "an authoritative and reliable source of what the software should do functionally" [Adzic01].

What Are Acceptance Tests?

Acceptance tests, as used in this book, are defined by the customer in collaboration with the developer and tested and created prior to implementation. They are not the traditional user acceptance tests [Cimperman01], which are performed after implementation "by the end user to determine if the system is working according to the specification in the contract." [Answers01] They are also not system tests which are usually independently written by testers by reading the requirements to ensure that the system meets those requirements. [Answers02] All three are related in that they are all black box tests—that is, they are independent of the implementation. It is the time and manner of creation in which they differ.

Where Are Acceptance Tests Used?

The concept of an acceptance test is defined by the intent of the test, not its implementation. You can apply an acceptance test at the unit, integration, or user interface level. You can use it as a validation test, which allows input to or produces outputs from an application installed in the customer's environment. Further, you can use it as a design verification test that ensures a unit or component meets it intended responsibility. In either case, the test makes certain the application is acceptable to the customer.

Who Creates the Acceptance Tests?

This book refers to a triad: the customer, developer, and tester. The power of three people working together [Crispin01] can create the bests acceptance tests.

If the triad writes the tests together, the distinction between user acceptance tests and system tests is practically eliminated. As will be shown, the three roles of customer, developer, and tester may be played by different individuals or by the same individual with different focuses.

What Types of Software Are Covered?

The acceptance tests covered in this book revolve mainly around requirements that have determinable results. These results are typical in business situations. You place an order, and the order total is determinable. On the other hand, you have a requirement to find the shortest possible path that goes through a number of points. For example, you want to determine the shortest driving trip that travels over every road in the United States. For a small number of roads (such as the interstate highways), the result is determinable by brute force. However, for a large number of roads, the answer is not determinable. You can have a test that checks the output of one way of solving the problem against the output of another way. But that does not guarantee that the shortest solution has been found.

How Will We Get to ATDD?

The answers to how and when the acceptance tests should be created are shown by a continuous example throughout this book. Each step in their creation and use is covered. Some books are devoted entirely to a single step and go into much greater detail than does this book. In particular, the references offer links for tools to automate the acceptance tests, to the agile process itself, to requirement elicitation, and to testing the other qualities of a software system (usability, performance, and so on).

The continuous example for Sam's CD Rental Store follows Sam's story in *Prefactoring—Extreme Abstraction, Extreme Separation, Extreme Readability*. That book used the tale as the context for examples of good design. *Prefactoring* covered some of the aspects of developer-customer interaction, because a good design requires understanding the customer's needs. *Prefactoring*'s focus was on the internal software quality. This book's focus is on externally visible quality. The two books complement each other.

Organization

The material is presented in six parts. The first part documents the tale of the triad members—customer, developer, tester—as they create a software system. It shows how acceptance testing permeates the entire process, from the project charter to individual stories. The second part covers details in acceptance testing, as simplification by separation. The third part explores general subjects, such as test presentation and valuation. The fourth part includes case studies from real-life situations. In some instances, the studies have been simplified to show only the relevant parts. The fifth part involves more technical issues, as how to handle test setup. The sixth part offers the appendices, which give additional information on topics as business value and test automation. For those who want to get the quick summary of ATDD and its benefits, read Chapter 25, "Retrospective and Perspective." Those who want to read the experiences of others, see the Epilogue.

Example Tables

The book presents tests with examples in tables rather than in narrative form. These tables follow the concepts of David Parnas, who states, "The tables constitute a precise requirements document" [Parnas01]. Some people prefer free text over tables. Those who prefer the narrative can easily convert tables to this form. The reverse is usually more difficult. Tables are familiar to spreadsheet users. Many business rules have conditions that are more easily tested with a table. From an analysis point of view, you can often find missing conditions by examining the values in a table's columns.

Automation After Communication

I emphasize acceptance tests as customer-developer-tester communication. If you don't have an acceptance test, you have nothing to automate. I do not advocate a particular test automation framework. When you automate an acceptance test that includes its accompanying requirement, you get what many term an executable specification [Melnik02], [Melnik03].

Acceptance tests can be manual. But if they are automated, you can use them as regression tests to ensure that future changes to the system do not affect previously implemented requirements. So the most effective use of the tests is as an executable specification. Appendix C, "Test Framework Examples," shows examples of test automation using several frameworks. The code for the examples is available online at http://atdd.biz.

ATDD Lineage

A Chinese proverb says, "There are many paths to the top of the mountain, but the view is always the same." And many of the paths share the same trail for portions of the journey. Although acceptance testing has been around for a long time, it was reinvigorated by extreme programming [Jefferies01]. Its manifestations include ATDD as described in this book, example-driven development (EDD) by Brian Marick [Marick01], behavior-driven development (BDD) by Dan North [Chelimsky01], story test-driven development (SDD) by Joshua Kerievsky of Industrial Logic [Kerievsky01], domain-driven design (DDD) by Eric Evans [Evans01], and executable acceptance test-driven development (EATDD) [EATDD01]. All these share the common goal of producing high-quality software. They aid developers and testers in understanding the customer's needs prior to implementation and customers being able to converse in their own domain language.

Many aspects are shared among the different approaches. ATDD in this book encompasses aspects of these other approaches. I've documented the parts that come specifically from the other driven developments (DDs), including Brian Marick's examples, Eric Evan's ubiquitous language, and Dan North's given-when-then template. The most visible differences are that the tests here are presented in table format rather than in a more textual format, such as BDD's Cucumber language, and they concentrate on functionality instead of the user interface. This book's version of ATDD matches closely that described by

Lasse Koskela [Koskela01] and Gojko Adzic [Adzic01] and follows the testing recommendations of Jim Coplien [Coplien01].

One of the most well-known DDs is test-driven development (TDD) by Kent Beck [Beck01]. TDD encompasses the developer's domain and tests the units or modules that comprise a system. TDD has the same quality goal as ATDD. The two interrelate because the acceptance tests can form a context in which to derive the tests for the units. TDD helps creates the best design for an application. A TDD design issue would be assigning responsibilities to particular modules or classes to pass all or part of an acceptance test.

Acceptance test driven development:
The answer is 42. Now implement it.

Summary

- Testable requirements have acceptance tests associated with them.
- ATDD involves developing requirement tests prior to implementation.
- ATDD can improve productivity.
- Acceptance tests are developed collaboratively between the customer, developer, and tester.

Chapter 4

An Introductory Acceptance Test

"If you don't know where you're going, you will wind up somewhere else."

Yogi Berra

An example of an acceptance test is presented, along with four ways that you can execute an acceptance test.

A Sample Business Rule

Here is an example from a previous project where Debbie and Tom created tests in collaboration with the customer. The business representative, Betty, presented the two of them with a business rule for giving discounts that she had obtained from one of the stakeholders. The stakeholder wanted to give discounts to the firm's customers based on what type of customer they were. Debbie had already completed implementing a previous requirement that determined the customer type. Here's the rule that Betty gave them:

> If Customer Type is Good and Item Total is less than or equal to $10.00,
>
> Then do not give a discount,
>
> Otherwise, give a 1% discount.
>
> If Customer Type is Excellent,
>
> Then give a discount of 1% for any order.
>
> If Item Total is greater than $50.00,
>
> Then give a discount of 5%.

This rule may seem clear. It uses consistent terms, such as *Customer Type* and *Item Total*. Debbie and Tom had previously gotten from Betty the definitions of those terms [Evans01]. For example, Item Total did not include taxes or shipping. But even with that consistency, there was an issue. Tom and Debbie looked at the rule and tried to figure out what the discount percentage should be if a customer who is good had an order total greater than $50.00. So Betty, Debbie, and Tom made up a table of examples.[1]

Discount Calculation		
Item Total	**Customer Rating**	**Discount Percentage?**
$10.00	Good	0%
$10.01	Good	1%
$50.01	Good	1% ??
$.01	Excellent	1%
$50.00	Excellent	1%
$50.01	Excellent	5%

The answers in this table of examples are going to be used to test the implementation. The first two rows show that the limit between giving a good customer no discount or a 1% discount is $10.00. The "less than or equal to" in the business rule is pretty clear. The tests just ensure that the implementation produced that result. The ?? was put after the 1 in the third example because it was unclear to the triad whether that was the right value. To what type of customer did the last statement in the rule apply?

The fourth row indicates that the discount for an excellent customer starts at the smallest possible Item Total. The fifth and sixth entries show that the discount increases just after the $50.00 point.[2]

Betty took this table back to the stakeholder. He looked it over and said that the interpretation was correct. He did not want to give a 5% discount to good customers. So ?? from that result was removed from that cell. There was now a set of tests that could be applied to the system. The correct discount amount test is not just a single case but includes cases for all possible combinations.

Tom suggested other possibilities. For example, what if Item Total was less than $0.00? Tom asked Betty whether this would ever happen. She said it might

1. See Appendix D, "Tables Everywhere," for an example of putting the rule into a table.

2. There could be even more interpretations of this business rule, as reviewers pointed out. For example, if Customer Rating is any other type than Good or Excellent, what should the discount be?

be possible, because Item Total could include a rebate coupon that was greater than the total of the items. So Tom added the following possibilities.

Discount Calculation		
Item Total	**Customer Rating**	**Discount Percentage?**
$–.01	Good	0%
$–.01	Excellent	1% ??

Tom explained that it didn't seem right to apply a discount percentage that would actually increase the amount that the customer owed. Based on this example, Betty went back to the stakeholder and confirmed that the percentage should be 0% if Item Total is less than 0 for any customer. So the table became as follows.

Discount Calculation		
Item Total	**Customer Rating**	**Discount Percentage?**
$–.01	Good	0%
$–.01	Excellent	0%

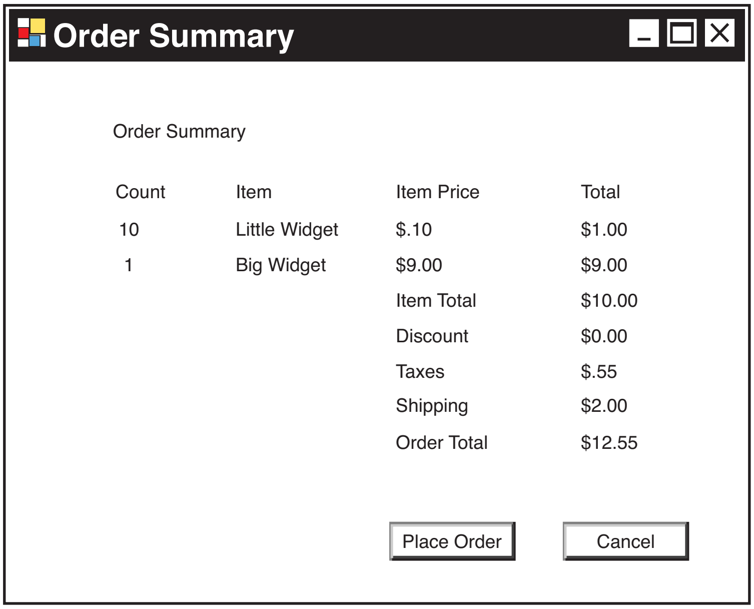
These examples were the acceptance tests for the system. If Debbie implemented these correctly, Betty would be satisfied. Now it was a matter of how Debbie and Tom were going to use these tests to test the system.

Implementing the Acceptance Tests

Tom and Debbie needed to apply these tests to the implementation they were developing. There were at least four possible ways to do this. First, Tom could create a test script that operates manually at the user interface level. Second, Debbie could create a test user interface that allows her or Tom to check the appropriate discount percentages. Third, Debbie could perform the tests using a unit testing framework. Fourth, Tom and Debbie could implement the tests with an acceptance test framework. Following are examples of how they could use each of these possibilities.

Test Script

In this case, the program has a user interface that allows a customer to enter an order. The user interface flow is much like Amazon or other order sites. The user enters an order and a summary screen appears, such as the one in Figure 4.1.



Order Summary

Count	Item	Item Price	Total
10	Little Widget	$.10	$1.00
1	Big Widget	$9.00	$9.00
		Item Total	$10.00
		Discount	$0.00
		Taxes	$.55
		Shipping	$2.00
		Order Total	$12.55

Figure 4.1 *Order Interface*

What Tom would have to do is to create a script that either he or Debbie would follow to test each of the six cases in the Discount Calculation table. He might start by computing what the actual discount amount should be for each case. Unless the Order Summary screen shows this percentage, this value is the only output Tom can check to ensure the calculation is correct. Here is an addition to the table that shows the amounts he needs to look for.

Discount Calculation				
Item Total	**Customer Rating**	**Discount Percentage?**	**Discount Amount?**	**Notes**
$10.00	Good	0%	$0.00	
$10.01	Good	1%	$0.10	Discount rounded down
$50.01	Good	1%	$0.50	Discount rounded down
$.01	Excellent	1%	$0.00	Discount rounded down
$50.00	Excellent	1%	$0.50	
$50.01	Excellent	5%	$2.50	Discount rounded down

The script would go something like this:

1. Log on as a customer who has the rating listed in the table.
2. Start an order, and put items in it until the total is the specified amount in the Item Total column on the test.
3. Check that the discount on the Order Summary screen matches Discount Amount in the table.

Then the test would be repeated five more times to cover all six cases. Either Tom or Debbie would do this once the discount feature and order features are implemented. This test should be run for all possible combinations. That would have been more difficult if there were more discount percentages for more customer types. There's another possible way to run these tests.

Test User Interface

To simplify executing the tests, Debbie could set up a user interface that connects to the discount calculation module in her code. This interface would be used only during testing. But having it would cut down on the work involved in showing that the percentage was correctly determined. The interface might be a command-line interface (CLI) or a graphical user interface (GUI). For example, a CLI might be this:

```
RunDiscountCalculatorTest  <item_total> <customer_type>
```

And when it is run for each case, such as

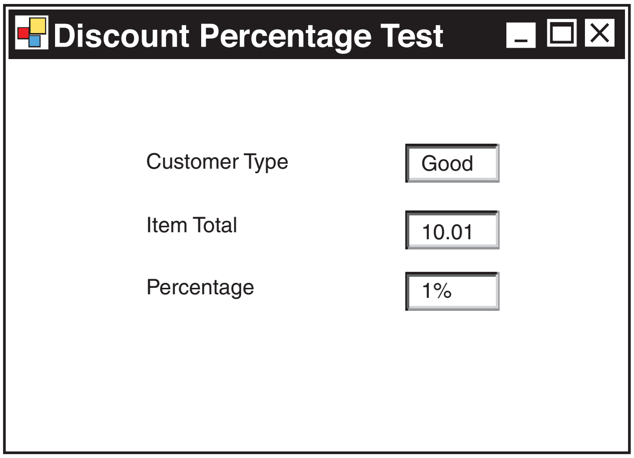
```
RunDiscountCalculatorTest 10,00 Good
```

It would output the result

```
0
```

A GUI, such as what's shown in Figure 4.2, might be connected to the CLI.

Regardless of whether it is a GUI or CLI, the user interface has penetrated into the system. It exposes a test point within the system that allows easier testing. Here's an analogy showing the differences between this method and Tom's original test script. Suppose you want to build a car that accelerates quickly. You know you need an engine that can increase its speed rapidly. If you could only check the engine operation as part of the car, you would need to put the engine in the car and then take the car on a test drive. If you had a test point for the engine speed inside the car, you could check how fast the engine sped up without driving the car. You could measure it in the garage. You'd save a lot of



Discount Percentage Test

Customer Type Good

Item Total 10.01

Percentage 1%

Figure 4.2 *User Interface for Testing*

time in on-the-road testing if the engine wasn't working properly. That doesn't mean you don't need to test the engine on the road. But if the engine isn't working by itself, you don't run the road test until the engine passes its own tests.

If you're not into cars, Figure 4.3 shows a context diagram. The Order Summary screen connects to the system through the standard user interface layer. The Discount Percentage user interface connects to some module inside the system. Let's call that module the Discount Calculator. By having a connection to the inside, a tester can check whether the internal behavior by itself is correct.

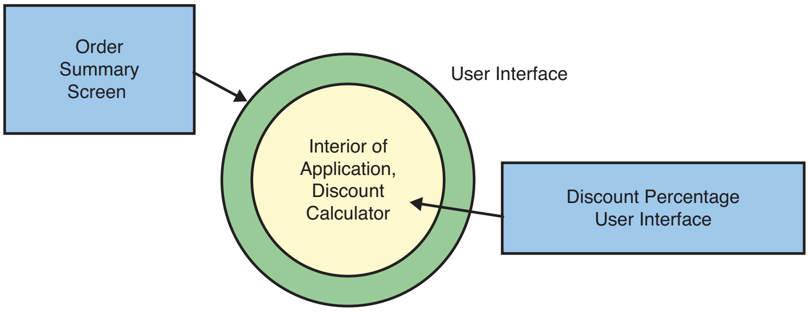

Order Summary Screen
User Interface
Interior of Application, Discount Calculator
Discount Percentage User Interface

Figure 4.3 *Context Diagram*

xUnit Test

The next way to perform the testing is to write the tests for the Discount Calculator in a unit testing framework. The framework used is usually in the language that the program is written in. There is a generic framework called xUnit that has versions for many programming languages. Here's a sample of what these tests look like in Java using Junit [Beck01]. The test would look similar in TestNG [Beust01], but the order of the parameters would be reversed:

```
class DiscountCalculatorTest  {
   @Test
   public void shouldCalculateDiscountPercentageForCustomer() {
      DiscountCalculator dc = new DiscountCalculator();
      assertEquals(0, dc.computeDiscountPercentage(10.0,
         Customer.Good));
      assertEquals(1, dc.computeDiscountPercentage (10.01,
         Customer.Good));
      assertEquals(1, dc.computeDiscountPercentage (50.01,
         Customer.Good));
      assertEquals(1, dc.computeDiscountPercentage(.01,
         Customer.Excellent));
      assertEquals(1, dc.computeDiscountPercentage(50.0,
         Customer.Excellent));
      assertEquals(5, dc.computeDiscountPercentage(50.01,
         Customer.Excellent));
   }
}
```

Any time there is a change in the examples that Betty and the stakeholder use to explain the business rule, Debbie may want these tests to conform to the changed examples. That's a bit of waste. The next testing framework can eliminate that waste.

Automated Acceptance Test

Betty, Debbie, and Tom agreed that the examples in the table accurately reflected the requirements and there would be less waste if the table did not have to be converted into another form for testing. Several available acceptance test frameworks use tables. Some examples are in Appendix C, "Test Framework Examples." With these frameworks, you describe the tests with a table similar to the one for the example.

The following test table works in table-based frameworks, such as the FitNesse and Fit frameworks. A similar style table can be used in narrative-form frameworks, such as Cucumber.[3] The table looks practically like the one that Betty presented to the stakeholder.

3. Fit is the Framework for Integrated Tests, developed by Ward Cunningham [Cunningham01], [Cunningham02]. Fit was incorporated into FitNesse by Bob Martin [Martin01]. Cucumber can be found in [Chelimsky01].

Discount Calculation		
Item Total	**Customer Rating**	**Discount Percentage()**
$10.00	Good	0%
$10.01	Good	1%
$50.01	Good	1%
$.01	Excellent	1%
$50.00	Excellent	1%
$50.01	Excellent	5%

Now when the table is used as a test, the Fit/FitNesse framework executes code that connects to the Discount Calculator. It gives the Discount Calculator the values in Item Total and Customer Rating. The Discount Calculator returns the Discount Percentage. The framework compares the returned value to the value in the table. If it agrees, the column shows up in green. If it does not, it shows up in red. The colors cannot be seen in this black-and-white book. So light gray represents green and dark gray represents red. The first time the test was run, the following table was output.

Discount Calculation		
Item Total	**Customer Rating**	**Discount Percentage()**
$10.00	Good	0%
$10.01	Good	1%
$50.01	Good	Expected 1% Actual 5%
$.01	Excellent	1%
$50.00	Excellent	1%
$50.01	Excellent	5%

With the results shown in the table, it was apparent there was an error in the Discount Calculator. Once it was fixed, Betty saw the passing tests as confirmation that the calculation was working as desired.

An Overall Test

If the discount test is applied using one of the last three forms, there still needs to be a test using the order interface. This ensures that processing an order is correctly connected to the Discount Calculator. The script for an order would be run for a couple of instances. But unless there was a large risk factor involved, the script might just be executed for a few cases, such as the following.

Discount Calculation			
Item Total	Customer Rating	Discount Percentage?	Discount Amount?
$10.01	Good	1%	$0.10
$50.01	Excellent	5%	$2.50

Testing Process

The acceptance test is the original table that Betty, Tom, and Debbie developed to clarify the business rule. This acceptance test can be used at four different levels, as described earlier in this chapter. Because the acceptance test was customer supplied, all four levels are considered acceptance tests in this book. The last two forms are automated by their nature. The second form—an interface to the Discount Calculator—can be automated. The test for an order could also be automated with a little more effort. However, you should still check it manually as well.

Passing the acceptance tests is necessary but insufficient to ensure that the system meets the customer needs. Other tests, such as those for quality attributes and usability (described in Chapter 3, "Testing Strategy"), also need to be passed. See [Meszaros02] for more information.

Summary

- Examples of requirements clarify the requirements.
- The examples can be used as tests for the implementation of the requirements.
- Tests for business rules can be executed in at least these four ways:
    - Creation through the user interface of a transaction that invokes the business rule
    - Development of a user interface that directly invokes the business rule
    - A unit test implemented in a language's unit testing framework
    - An automated test that communicates with the business rule module

This page intentionally left blank

Index

A

acceptance criteria
  defined, 12
  explained, 95
  feature acceptance criteria, 45-46
  story acceptance criteria, 52
acceptance tests
  acceptance criteria. *See* acceptance criteria
  agile principles, 20-21
  anti-missile acceptance test, 276
  automated acceptance tests, 35-36, 89-90
  Bad Customer ID test, 90
  benefits, 2, 214-215
  blocks
    change issues, 213
    explained, 212
    monads, 212
    risks, 214
    unavailable customers, 213
  breaking up stories with, 95-96
  as bug reports, 201-202
    production bugs, 202
    regression testing, 202
    root cause analysis, 201-202
  business capabilities, 172-178
  business rules. *See* business rules
  Card Processor Charges Matches Rental Charges test, 105
  case studies. *See* case studies
  CD Already Rented test, 84
  CD Limit Reached and Late Rental test, 88
  CD Limit Reached test, 85-86
  Charge Submitted During Check-In test, 108
  Charges Agree with Transfer test, 104
  Check-In CD test
    business rules, 100
    creating, 101
    Cucumber framework, 297
    Fit framework, 286-287
    relationships, 165
    Robot framework, 295-296
    Slim—Cucumber Style framework, 291
    Slim—Table Style framework, 288-290
    story map of activities, 113-114
  Check-Out CD test
    Cucumber framework, 296-297
    Fit framework, 284-285
    Robot framework, 295
    Slim—Cucumber Style framework, 291
    Slim—Table Style framework, 288-289
  communication, 66-68, 212
  context. *See* context
  control and observation points, 27
  creating, 69-70
  Current Late Rental When Renting test, 86
  customer examples
    acceptance test detail, 275
    explained, 274
    fuzzy acceptance tests, 274
  data conversion projects, 170
  database conversion projects, 170
  decoupling user interface
    explained, 131-133
    separating user interface from service, 157-158
    simplifying testing by, 136
  defined, 2-3, 12

developer acceptance tests
  field display tests, 145-147
  tabular display tests, 147-151
development review, 115-116
  from charter to deliverable, 120-121
  deployment plans, 120
  quality attribute tests, 117-119
  separating state from display, 116-117
  usability testing, 116
  workflow tests, 119-120
device testing, 254-255
events, 137-139
explained, 3, 211-212
external interfaces
  component tests, 107-111
  creating acceptance tests for, 107
  explained, 100-108
  system context, 112-113
  test doubles and mocks, 111-112
feedback, 18
importance of, 10-11
internal design from tests, 252-254
Invalid Card Response Is Charge Declined test, 110
lack of, 170-173
large systems, triads for, 167-169
lean principles, 20-21
legacy systems, 172
manual testing, 172
multi-level tests, 90-93
objectives, checking, 93-94
origin and development, 5-6
positive and negative testing, 26-27
post-implementation tests, 17-18
power of three, 277
preimplementation tests, 19-20
process, 210
project charters, 39-42
relationships
  alternative representations, 167
  entities and, 161-163
  explained, 161
  multiple relationships, 163-165
requirements
  constraints, 275
  decoupling, 276
  defined, 275
  documenting, 275-276
  explained, 27-28
  high-level requirements, 42-46
  separation of issues, 276-277
  user interfaces as part of requirements process, 255
risks, 214
role in overall development process, 209-212
running, 25-26, 78-80
scenarios, 57
  conceptual flow, 65
  scenario handling, 178-179
  story maps, 63-64
separation of issues. *See* separation of issues
service provider tests
  comparing implementations, 155-156
  explained, 151
  interface, 151-154
  quality attribute tests, 155
states
  internal states, 142-144
  state tables, 139-142
  state transitions, 139-142
  transient or persistent states, 144
stories
  acceptance criteria, 52
  breaking down features into stories, 48
  cross-story issues, 88
  customer terms, 54
  developing, 51
  explained, 47-48
  INVEST criteria, 55-56
  persona, 50-51
  role attributes, 49-50
  roles, 49
  size of, 53-54
  story estimates, 52
  story maps, 63-64, 113-114
  use cases, 57-63
success stories, 324-332
tables with, 298-300
test evaluation
  confirming environment, 193

fragility, 192-194
idempotence, 192
implementation issues, 197
points to remember, 197-198
separation of concerns, 194-195
spell checking, 192
test failure, 195
test redundancy, 196-197
test sequence, 193-194
understandability to customers, 191-192
test exercises
calculator test exercise, 310-313
Sam's CD Rental exercise, 314
triangle exercise, 314
test frameworks
Cucumber, 296
explained, 283
Fit, 284
Robot, 295
Slim—Cucumber Style framework, 291
Slim—Table Style, 286-284
websites, 298
test platforms, 251-252
test presentation
complex data, 187
custom table forms, 188
customer understood tables, 183-185
specifying multiple actions, 185-187
tables versus text, 185
test scripts, 31-33
test setup
ameliorating potential problems, 259-260
common setup, 257-259
persistent storage issues, 260-261
test order, 260
test structure
action tables, 75-76
calculation tables, 73-74
data tables, 74-75
explained, 71-73
testing layers, 210-211
testing matrix, 23-24
testing process, 37
testing systems with random events, 277
types of tests, 23-25
tests for business rules, 86
tests for exception scenarios, 83-87
tests with example values, 76-78
tests with values in text, 78
user interface tests, 33-34, 93
uses of
breaking down stories, 200
degree of doneness, 199
developer stories, 200
estimation aid, 200
workflows, improving, 178-181
xUnit tests, 34-35
action tables, 75-76
activities, story map of, 113-114
Acuminous Ltd., 326
Adzic, Gojko, 6
"The Agile Manifesto," 20
agile principles, 20-21
agileartisans.com, 327
airline industry
conversion to SHARES reservation system, 252
ubiquitous language, 207
Allowed to Reserve business rule, 126
separation of issues, 126-127
simplified rule, 128
allowedToReserve() method, 253-254
ameliorating potential problems, 259-260
Angelou, Maya, 83
anti-missile acceptance test, 276
applications, determining applications to run on server, 244-248
assigning business value, 280-281
Atwood, Margaret, 1
auto mileage displays, 18
automated acceptance tests, 35-36, 89-90

B

Bad Customer ID test, 90
Bain, Scott, 27
BDD (behavior-driven development), 5
Beck, Kent, 6, 305
behavior-driven development (BDD), 5
benefits to acceptance testing, 2, 214-215
Berra, Yogi, 29
big picture test, 227-228

black box testing, 255-256
blocks to acceptance testing
  change issues, 213
  explained, 212
  monads, 212
  risks, 214
  unavailable customers, 213
boundary value analysis, 255
breaking down tests, 264-269
  disallowed domain tests, 268-269
  domain breakdown tests, 266-268
  local-part validation, 265-266
  tests to ensure connection, 269
  verification tests, 269
breaking up stories with acceptance tests, 95-96, 200
Browning, Elizabeth Barrett, 23
Buffett, Warren, 172
bug reports
  acceptance tests as, 201-202
  production bugs, 202
  regression testing, 202
  root cause analysis, 201-202
Burger King, off-menu ordering, 178-179
business capabilities, 172-178
business rules
  business rule tests, 96-100
  Check-In CD test, 100
  example of, 29-31
  exposing, 178
  reusable business rules, 158-159
  separation of issues, 125-128
  tests for, 86
business value
  business value charts, 281
  developer stories, 281-282
  estimating, 280-281
  explained, 279-280
  sources of, 279
  tracking, 223

C

calculation tables, 73-74
calculator test exercise, 310-313
Cann, Richard, 229
Cannan, Dawn, 324
Card Processor Charges Matches Rental Charges test, 105
Carroll, Lewis, 6
case studies
  e-mail addresses case study, 263-264
    breaking down tests, 264-269
    disallowed domain tests, 268-269
    domain tests, 266-268
    local-part validation, 265-266
    tests to ensure connection, 269
    verification tests, 269
  highly available platform
    context for switching servers, 243
    test for switching servers, 244
    test for technical rule, 246-248
  library print server case study
    context, 235-236
    workflow test, 236-241
  retirement contributions case study
    big picture test, 227-228
    business value tracking, 223
    context, 219-220
    event table, 228
    exception: discrepancies less than a dollar, 223-224
    exception: missing participant, 225
    main course test, 220-222
    state transitions, 228-230
    two simultaneous exceptions, 225-227
  signal processing case study
    context, 229
    developer tests, 233
    sound levels, 229-232
case tests, 255
category-based rental fees
  Cucumber framework, 296-297
  Fit framework, 287
  Robot framework, 295-296
  Slim—Cucumber Style framework, 291
  Slim—Table Style framework, 287-291
CD Already Rented test, 84
CD Limit Reached and Late Rental test, 88
CD Limit Reached test, 85-86

CD rental example
  action tables, 75-76
  Allowed to Reserve business rule, 126
    separation of issues, 126-127
    simplified rule, 128
  allowedToReserve() method, 253-254
  Bad Customer ID test, 90
  calculation tables, 73-74
  Card Processor Charges Matches Rental Charges test, 105
  CD Already Rented test, 84
  CD Limit Reached and Late Rental test, 88
  CD Limit Reached test, 85-86
  CD rental limit reached, 185-187
  Charge Submitted During Check-In test, 108
  Charges Agree with Transfer test, 104
  Check-In CD test
    business rules, 100
    creating, 101
    Cucumber framework, 297
    Fit framework, 286-287
    relationships, 165
    Robot framework, 295-296
    Slim—Cucumber Style framework, 291
    story map of activities, 113-114
  Check-Out CD test
    creating, 76-78
    Cucumber framework, 296-297
    Fit framework, 284-285
    Robot framework, 295
    Slim—Cucumber Style framework, 291
    Slim—Table Style framework, 288-289
  Check-Out screen, 93
  Current Late Rental When Renting test, 86
  customer terms, 54
  customer understood tables, 183-185
  data tables, 74-75
  development review, 115-116
    from charter to deliverable, 120-121
    deployment plans, 120
    quality attribute tests, 117-119
    separating state from display, 116-117
    usability testing, 116
  discount calculation
    automated acceptance tests, 35-36
    content diagram, 34
    explained, 29-31
    overall test, 36-37
    test script, 35-36
    xUnit test, 34-35
  display for reservation allowed, 132-134
  events, 137-139
  exercises, 314
  feature acceptance criteria, 45
  features list, 44
  field display tests, 145-147
  Invalid Card Response Is Charge Declined test, 110
  people, 13
  persona, 51
  post-implementation tests, 17-18
  preimplementation tests, 19-20
  relationships
    alternative representations, 167
    entities and, 161-163
    explained, 161
    multiple relationships, 163-165
  rental counts, computing, 130
  rental fees. *See* rental fees
  rental history, 128-130
  rental sequence, 113
  role attributes, 50
  roles, 49
  service provider tests
    comparing implementations, 155-156
    explained, 153
    interface, 153-154
    quality attribute tests, 155
  states
    internal states, 142-144
    state tables, 139-142
    state transitions, 139-142
    transient or persistent states, 144
  stories, 51
  story acceptance criteria, 52
  Submit Charge story, 100-108

system, 11-13
tabular display tests, 147-151
use cases from user stories, 57-63
ZIP code lookup, 151-156
change issues, 213
Charge Rentals story, 95
Charge Submitted During Check-In test, 108
Charges Agree with Transfer test, 104
charters, 39-42
charts, business value charts, 281
Check-In CD test
business rules, 100
creating, 101
Cucumber framework, 297
Fit framework, 286-287
relationships, 165
Robot framework, 295-296
Slim—Cucumber Style framework, 291
Slim—Table Style framework, 288-290
story map of activities, 113-114
checking
objectives, 93-94
spelling, 192
Check-Out CD test
creating, 76-78
Cucumber framework, 296-297
Fit framework, 284-285
Robot framework, 295
Slim—Cucumber Style framework, 291
Slim—Table Style framework, 288-289
Check-Out screen, 93
Cicero, 131
Codebetter.com, 330
Cohen, Leonard, 183
Cohn, Mike, 200
common test setup, 257-259
communication, 66-68, 212
comparing implementations, 155-156
complex data, 187
component tests
explained, 25, 211
for external interfaces, 107-111
conceptual flow, 65
confirming environment, 193
connections, tests to ensure connection, 269
Constantine, Larry, 276
constraints, 263, 275
context
context diagrams, 70
e-mail addresses case study, 263-264
explained, 70, 273-274
highly available platform, 243
library print server case study, 235-236
retirement contributions case study, 219-220
signal processing case study, 229
system context, 112-113
continuity, 207
control points, 27
conversion projects
currency conversion example
acceptance test approach, 307-310
test-driven approach, 306-307
unit tests, 306-307
data conversion, 170
database conversion, 170
Cooper, Ian, 330
Coplien, Jim, 6
copying files, 314
Cresswell, Stephen, 326
cross-story issues, 88
Cucumber framework, 296
currency conversion example
context, 305
test-driven approach, 306-307
unit tests, 306-307
Current Late Rental When Renting test, 86
custom table forms, 188
customers
customer examples
acceptance test detail, 275
explained, 274
fuzzy acceptance tests, 274
customer terms, 54
customer understood tables, 183-185
customer understood tests, 191-192, 211
customer unit, 16
data tables, 187
unavailable customers, 213

D

data conversion projects, 170
data tables, 74-75, 304
database conversion projects, 170
DDD (domain-driven design), 5
  overlapping domains, 207
  ubiquitous language, 205-207
decision table testing, 256
decoupling user interface
  decoupling requirements, 276
  explained, 131
  separating user interface from service, 157-158
  simplifying testing by, 136
degree of doneness, 199
delta, 260
deployment plans, 120
determining applications to run on server, 244-248
developer acceptance tests
  field display tests, 145-147
  signal processing case study, 233
  tabular display tests, 147-151
developer stories, 200, 281-282
developer unit, 16
development of acceptance testing, 5-6
development process
  explained, 9-10
  role of acceptance tests in, 209-212
development review, 115-116
  from charter to deliverable, 120-121
  deployment plans, 120
  quality attribute tests, 117-119
  separating state from display, 116-117
  usability testing, 116
  workflow tests, 119-120
device testing, 254-255
Dijkstra, Edsger, 191
disallowed domain tests, 268-269
discount business rules, 299-301
discount calculation
  automated acceptance tests, 35-36
  content diagram, 34
  explained, 29-31
  overall test, 36-37
  test script, 31-33
  xUnit test, 34-35
discrepancies, handling in retirement contributions case study, 223-224
documenting
  bugs with acceptance tests, 201-202
  requirements, 275-276
  with use cases, 63
documents, printing to print queue (workflow), 237-238
domain tests
  disallowed domain tests, 268-269
  domain breakdown tests, 266-268
domain-driven design. *See* DDD (domain-driven design)
domains, overlapping, 207
doneness, degree of, 199
DRY (Don't Repeat Yourself) principle, 259
Dyer, Wayne, 100

E

EATDD (executable acceptance test-driven development), 5
EDD (example-driven development), 5
e-mail addresses case study
  breaking down tests, 264-269
  context, 263-264
  disallowed domain tests, 268-269
  domain tests, 266-268
  local-part validation, 265-266
  tests to ensure connection, 269
  verification tests, 269
embedded technical projects, 283
ensuring connections, 269
entities and relationships, 161-163
environment, confirming, 193
EnvisionWare, 235
equivalence partitioning, 255
estimates (story), 52, 55
estimating business value, 280-281
estimation aid, 200
evaluating tests. *See* test evaluation
Evans, Eric, 5
events
  explained, 137-139
  retirement contributions case study, 220, 228
example-driven development (EDD), 5

example values, tests with, 76-78
exceptions
  retirement contributions case study
    discrepancies less than a dollar, 223-224
    missing participant, 225
    two simultaneous exceptions, 225-227
  scenario handling, 178-179
  tests for exception scenarios, 83-87
executable acceptance test-driven development (EATDD), 5
exercises
  calculator test exercise, 311-313
  file copying exercise, 314
  Sam's CD Rental exercise, 314
  triangle exercise, 314
expected output, 220
exploratory tests, 24
exposing business rules for testing, 178
external constraints, 275
external interfaces, 100-108
  component tests, 108-111
  creating acceptance tests for, 107
  system context, 112-113
  test doubles and mocks, 111-112

F

failure of tests, 195
Feather, Michael, 172
feature acceptance criteria, 45-46
features list
  breaking down into stories, 48
  explained, 42-44
feedback, 18
field display tests, 145-147
file copying exercise, 314
Fit framework
  category-based rental fees, 287
  Check-In CD, 286-287
  Check-Out CD, 284-285
  explained, 284
  setup, 284
FitNesse, 276
forms, custom table forms, 188
fragility of tests, 192-194
frameworks. *See* test frameworks
fuzzy acceptance tests, 274

G-H

Gaye, Marvin, 299
Grozier Technical Systems, 229

headers for Slim—Table Style framework, 288
high-level requirements
  explained, 42-46
  feature acceptance criteria, 45-46
  features list, 42-44
highly available platform
  context for switching servers, 243
  test for switching servers, 244
    send alert to administrator, 245-246
    server goes down, 245
  test for technical rule, 244-248
Hofstadter, Douglas, 167, 273
Honda Insight charter objectives, 41
Hunter, Ivy, 299

I

idempotence, 192
identity, 206
implementation issues
  comparing implementations, 155-156
  retirement contributions case study, 222
  and test evaluation, 197
importance of acceptance testing, 10-11
improving workflows, 178-181
independence of stories, 55
internal constraints, 275
internal design from tests, 252-254
internal states, 142-144
Invalid Card Response Is Charge Declined test, 110
INVEST criteria, 55-56
issues, separation of, 194-195, 276-277

J-K

Johnson, Samuel, 95

Kennedy, Robert, 209
Kerievsky, Joshua, 5
King, Carole, 243
Knuth, Donald, 263

Kolsky, Amir, 27, 194
Koskela, Lasse, 6

L

lack of acceptance tests, 170-173
language, ubiquitous, 205-207
large systems, triads for, 167-169
Le Van, Gabriel, 325
lean principles, 20-21
legacy systems, 172
Leq (equivalent continuous sound level), 229-232
library print server case study
  context, 235-236
  workflow test, 236-241
    workflow for printing jobs from print queue, 239-241
    workflow of printing two documents to print queue, 237-238
local-part validation, 265-266
Lovasik, Lawrence G., 199

M

main course test (retirement contributions case study)
  events, 220
  expected output, 220
  implementation issues, 222
  separation of concerns, 222
  setup, 220-221
manifestations, 275
manual testing, 172
Marcos, Gary, 332
Marick, Brian, 5
Martin, Micah, 276
Martin, Robert, 276
Marx, Groucho, 67, 305
Maslow, Abraham, 299
McDonald's, off-menu ordering, 179
McNelly, Greg, 331
measurability of objectives, 43
Meszaros, Gerard, 23, 259
methods, allowedToReserve(), 253-254
mileage displays in autos, 18
missing participants, handling in retirement contributions case study, 225
mocks, 111-112
Model-View-Controller pattern, 253
module tests, 211
monads, 212
money
  currency conversion example
    acceptance test approach, 307-310
    context, 305
    test-driven approach, 306-307
  problems with, 305
multi-level tests, 90-93
multiple actions, specifying, 185-187
multiple relationships, 163-165
Murray, Bill, 14

N

negative testing, 26-27
negotiability of stories, 55
Newbury, Mickey, 137
North, Dan, 5

O

objectives
  checking, 93-94
  of project charters, 40-42
observation points, 27
Once and Only Once Principle, 259
order of tests, 260
overlapping domains, 207

P

partitions, equivalence partitioning, 255
passionatetester.com, 324
Patanjali, 39
patterns, Model-View-Controller, 253
persistent states, 144
persistent storage issues, 260-261
persona, 50-51
platforms (test), 251-252
Poppendieck, Mary, 20
Poppendieck, Tom, 20
positive testing, 26-27
post-implementation tests, 17-18
preimplementation tests, 19-20

print queue
  printing jobs from (workflow), 239-241
  printing multiple documents to (workflow), 237-238
process, 210
production bugs, 202
Progressive Insurance, 330-331
project charters, 39-42

Q-R

quality attribute requirements, 303-304
quality attribute tests, 117-119, 155

random events, testing systems with, 277
redundancy of tests, 196-197
regression testing, 202
Reinertsen, Don, 11
relationships
  alternative representations, 167
  entities and, 161-163
  explained, 161
  multiple relationships, 163-165
relative results, 260
relative story placement, 280
relative story values, 280
rental counts, computing, 129-130
rental fees
  business rule tests, 96-100
  category-based. *See* category-based rental fees
  rental rates table, 304
  table of, 97-99
  tables versus text, 185
rental history, 128-130
rental sequence, 113
requirements
  constraints, 275
  decoupling, 276
  defined, 275
  documenting, 275-276
  explained, 27-28
  high-level requirements, 42-46
    feature acceptance criteria, 45-46
    features list, 42-44
  quality attribute requirements, 303-304
  requirements tables, 300-303
  separation of issues, 276-277
  user interfaces as part of requirements process, 255
results, relative, 260
retirement contributions case study
  big picture test, 227-228
  business value tracking, 223
  context, 219-220
  event table, 228
  exception: discrepancies less than a dollar, 223-224
  exception: missing participant, 225
  main course test
    events, 220
    expected output, 220
    implementation issues, 222
    separation of concerns, 222
    setup, 220-221
  state transitions, 228-230
  two simultaneous exceptions, 225-227
reusable business rules, 158-159
Richardson, Jared, 327
risks, 214
Robot framework, 295
  category-based rental fees, 296
  Check-In CD test, 296
  Check-Out CD test, 295
  setup, 295
roles
  explained, 49
  role attributes, 49-50
root cause analysis, 201-202
rules. *See* business rules
running tests, 25-26, 78

S

Sabre reservation system, 252
Sam's CD Rental. *See* CD rental example
Sandburg, Carl, 125, 310
Satir, Virginia, 213
scenarios, 57
  conceptual flow, 65
  Slim—Cucumber Style framework scenario library, 291-294
  story maps, 63-64
  tests for exception scenarios, 83-87
  use cases from user stories, 57-63

SDD (story test-driven development), 5
separation of issues, 194-195, 276-277
  Allowed to Reserve business rule example, 125-128
  rental history example, 128-130
  retirement contributions case study, 222
  separating user interface from service, 157-158
sequence of tests, 193-194
servers, switching
  context, 243
  determining applications to run on server, 244-248
  test for, 244
    send alert to administrator, 246-244
    server goes down, 245
service provider tests
  comparing implementations, 155-156
  explained, 151
  interface, 151-154
  quality attribute tests, 155
services, separating user interface from, 157-158
setup
  Fit framework, 284
  retirement contributions case study, 220-221
  Robot framework, 295
  Slim—Cucumber Style framework, 291-292
  Slim—Table Style framework, 287-291
Shalloway's Law, 259
SHARES reservation system, 252
Shaw, George Bernard, 205
Shore, Jim, 42
signal processing case study
  context, 229
  developer tests, 233
  sound levels, 229-232
simultaneous exceptions, 225-227
size of stories, 53-56
Slim—Cucumber Style framework, 291
  category-based rental fees, 291
  Check-In CD test, 291
  Check-Out CD test, 291
  scenario library, 291-294
  setup, 291
Slim—Table Style framework, 287-291
  category-based rental fees, 291
  Check-In CD test, 288-290
  Check-Out CD test, 288-289
  header, 288
  setup, 288
SMART, 40
software development, 9-10
sound levels, signal processing case study, 229-232
specifying multiple actions, 185-187
speed of feedback, 18
spell checking, 192
states
  internal states, 142-144
  separating state from display, 116-117
  state tables, 139-142
  state transitions, 139-142
    retirement contributions case study, 228-230
    state transition testing, 255
  transient or persistent states, 144
Stevenson, William, 299
stories
  acceptance criteria, 52
  breaking down features into stories, 48
  breaking up with acceptance tests, 95-96, 200
  business rules. *See* business rules
  business value. *See* business value
  cross-story issues, 88
  customer terms, 54
  developer stories, 200, 281-282
  developing, 51
  explained, 47-48
  INVEST criteria, 55-56
  persona, 50-51
  relative story placement, 280
  relative story values, 280
  roles
    explained, 49
    role attributes, 49-50
  size of, 53-54
  story estimates, 52

story maps, 63-64, 113-114
use cases, 57-63
story maps, 63-64, 113-114
story test-driven development (SDD), 5
Submit Charge story, 100-108
success stories, 324-332
switching servers
context, 243
determining applications to run on server, 244-248
test for, 244
send alert to administrator, 245-246
server goes down, 245

T

tables
action tables, 75-76
calculation tables, 73-74
customer understood tables, 183-185
data tables, 74-75, 304
event tables, 137-139
quality attribute requirements, 303-304
requirements tables, 300-303
state tables, 139-142
tabular display tests, 147-151
versus text, 78, 185
user interface tests with, 299-300
tabular display tests, 147-151
TDD (test-driven development), 6
technical projects, 283
technical rules, test for, 244-248
test doubles, 111-112
test-driven approach, 306-307
Test-Driven Development by Example (Beck), 305
test-driven development (TDD), 6
test evaluation
confirming environment, 193
fragility, 192-194
idempotence, 192
implementation issues, 197
points to remember, 197-198
separation of concerns, 194-195
spell checking, 192
test failure, 195
test redundancy, 196-197
test sequence, 193-194
understandability to customers, 191-192
test failure, 195
test frameworks
Cucumber, 296
explained, 283
Fit, 284
category-based rental fees, 287
Check-In CD, 286-287
Check-Out CD, 284-285
setup, 284
Robot, 295
category-based rental fees, 296
Check-In CD test, 296
Check-Out CD test, 295
setup, 295
Slim—Cucumber Style framework, 291
category-based rental fees, 291
Check-In CD test, 291
Check-Out CD test, 291
scenario library, 291-294
setup, 291
Slim—Table Style, 287-291
Check-In CD test, 288-290
Check-Out CD test, 288-289
header, 288
setup, 288
Slim—Table Style framework, 287-291
websites, 298
test platforms, 248-252
test presentation
complex data, 187
custom table forms, 188
customer understood tables, 183-185
specifying multiple actions, 185-187
tables versus text, 185
test redundancy, 196-197
test scripts, 31-33
test sequence, 193-194
test setup
ameliorating potential problems, 259-260
common setup, 257-259
persistent storage issues, 260-261
test order, 260
testability of stories, 57

testing layers, 210-211
testing matrix, 23-24
testing systems with random events, 277
testing unit, 16
tests
  acceptance tests. *See* acceptance tests
  black box testing, 255-256
  component tests, 25, 211
  manual testing, 172
  modules tests, 211
  post-implementation tests, 17-18
  preimplementation tests, 19-20
  quality attribute tests, 117-119
  regression testing, 202
  unit tests
    explained, 25, 211, 256
    limitations, 88
  workflow tests, 119-120
  xUnit tests, 34-35
text, versus tables, 78, 185
time periods, calculating, 99
tracking business value, 223
transient states, 144
triads
  explained, 15-16
  triads for large systems, 167-169
triangle exercise, 314
types of tests, 23-25

U

ubiquitous language, 205-207
unavailable customers, 213
unit tests
  explained, 25, 211, 256
  limitations, 88
usability testing, 24, 116
use cases from user stories, 57-63
user interfaces
  decoupling. *See* decoupling user interface
  external interfaces
    creating acceptance tests for, 107
    explained, 100-108
    system context, 112-113
  as part of requirements process, 255
  user interface tests, 33-34, 93
uses of acceptance tests
  breaking down stories, 200
  degree of doneness, 199
  developer stories, 200
  estimation aid, 200

V

validation, local-part, 265-266
value of stories, 55
verification tests, 269
Visual Hospital Touchscreen Solution, 325

W-X-Y-Z

Wake, Bill, 55
Walsh, Rob, 235
websites, test framework websites, 298
Weinberg, Jerry, 277
West, Mae, 257
Wilde, Oscar, 161
workflows
  improving, 178-181
  workflow tests, 119-120
    library print server case study, 236-241
    test sequence, 193-194

xUnit tests, 34-35

ZIP code lookup, 151-156