

The Addison-Wesley Signature Series

AGILE TESTING

A PRACTICAL GUIDE FOR
TESTERS AND AGILE TEAMS

LISA CRISPIN
JANET GREGORY



Forewords by Mike Cohn and Brian Marick

A MIKE COHN
SIGNATURE
BOOK
Mike Cohn



Praise for *Agile Testing*

“As Agile methods have entered the mainstream, we’ve learned a lot about how the testing discipline fits into Agile projects. Lisa and Janet give us a solid look at what to do, and what to avoid, in Agile testing.”

—Ron Jeffries, www.XProgramming.com

“An excellent introduction to agile and how it affects the software test community!”

—Gerard Meszaros, Agile Practice Lead and Chief Test Strategist at Solution Frameworks, Inc., an agile coaching and lean software development consultancy

“In sports and music, people know the importance of practicing technique until it becomes a part of the way they do things. This book is about some of the most fundamental techniques in software development—how to build quality into code—techniques that should become second nature to every development team. The book provides both broad and in-depth coverage of how to move testing to the front of the development process, along with a liberal sprinkling of real-life examples that bring the book to life.”

—Mary Poppendieck, Author of *Lean Software Development* and *Implementing Lean Software Development*

“Refreshingly pragmatic. Chock-full of wisdom. Absent of dogma. This book is a game-changer. Every software professional should read it.”

—Uncle Bob Martin, Object Mentor, Inc.

“With *Agile Testing*, Lisa and Janet have used their holistic sensibility of testing to describe a culture shift for testers and teams willing to elevate their test effectiveness. The combination of real-life project experiences and specific techniques provide an excellent way to learn and adapt to continually changing project needs.”


—Adam Geras, M.Sc. Developer-Tester, Ideaca Knowledge Services

“On Agile projects, everyone seems to ask, ‘But, what about testing?’ Is it the development team’s responsibility entirely, the testing team, or a collaborative effort between developers and testers? Or, ‘How much testing should we automate?’ Lisa and Janet have written a book that finally answers these types of questions and more! Whether you’re a tester, developer, or manager, you’ll learn many great examples and stories from the real-world work experiences they’ve shared in this excellent book.”

—Paul Duvall, CTO of Stelligent and co-author of *Continuous Integration: Improving Software Quality and Reducing Risk*

“Finally a book for testers on Agile teams that acknowledges there is not just one right way! *Agile Testing* provides comprehensive coverage of the issues testers face when they move to Agile: from tools and metrics to roles and process. Illustrated with numerous stories and examples from many contributors, it gives a clear picture of what successful Agile testers are doing today.”

—Bret Pettichord, Chief Technical Officer of WatirCraft and Lead Developer of Watir



Story PA-2
As an internet shopper on LotsO'Stuff.xx, I want free shipping when my order exceeds the free shipping threshold, so that I can take advantage of ordering more at one time.

Figure 8-1 Story to set up conversation

given a wordy requirements document that includes every detail of the feature set. On an agile project, the customer team and development team strike up a conversation based on the story. The team needs requirements of some kind, and they need them at a level that will let them start writing working code almost immediately. To do this, we need examples to turn into tests that will confirm what the customer really wants.

These business-facing tests address business requirements. These tests help provide the big picture and enough details to guide coding. Business-facing tests express requirements based on examples and use a language and format that both the customer and development teams can understand. Examples form the basis of learning the desired behavior of each feature, and we use those examples as the basis for our story tests in Quadrants 2 (see Figure 8-2).

Business-facing tests are also called “customer-facing,” “story,” “customer,” and “acceptance” tests. The term “acceptance test” is particularly confusing, because it makes some people think only of “user acceptance tests.” In the context of agile development, acceptance tests generally refer to the business-facing tests, but the term could also include the technology-facing tests from Quadrant 4, such as the customer’s criteria for system performance or security. In this chapter, we’re discussing only the business-facing tests that support the team by guiding development and providing quick feedback.

As we explained in the previous two chapters, the order in which we present these four quadrants isn’t related to the order in which we might perform

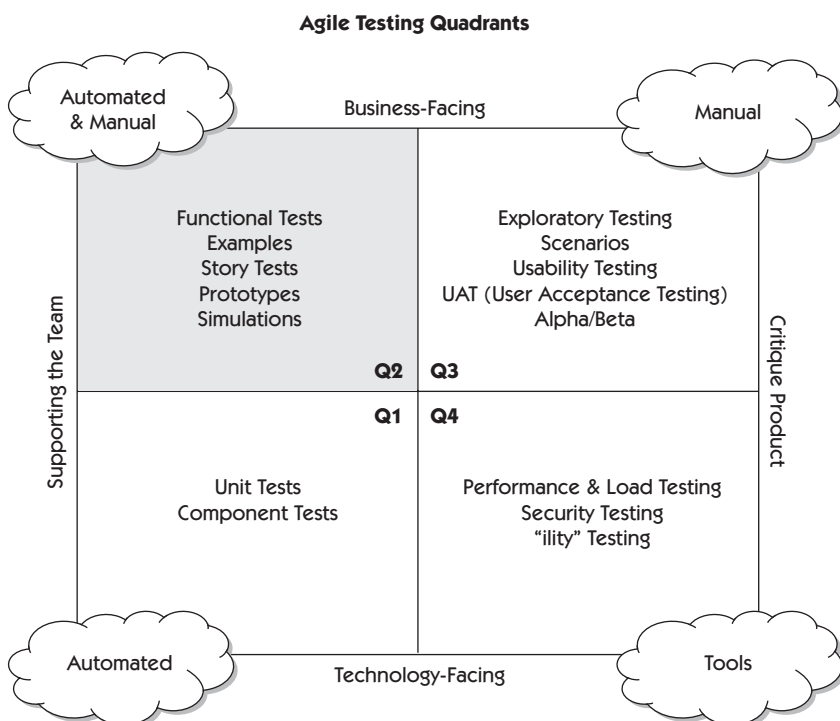


Figure 8-2 The Agile Testing Quadrants, highlighting Quadrant 2

Part V, "An Iteration in the Life," examines the order in which we perform tests from the different quadrants.

activities from each quadrant. The business-facing tests in Quadrant 2 are written for each story before coding is started, because they help the team understand what code to write. Like the tests in Quadrant 1, these tests drive development, but at a higher level. Quadrant 1 activities ensure internal quality, maximize team productivity, and minimize technical debt. Quadrant 2 tests define and verify external quality, and help us know when we're done.

The customer tests to drive coding are generally written in an executable format, and automated, so that team members can run the tests as often as they like in order to see if the functionality works as desired. These tests, or some subset of them, will become part of an automated regression suite so that future development doesn't unintentionally change system behavior.

As we discuss the stories and examples of desired behavior, we must also define nonfunctional requirements such as performance, security, and usability. We'll

also make note of scenarios for manual exploratory testing. We'll talk about these other types of testing activities in the chapters on Quadrants 3 and 4.

We hear lots of questions relating to how agile teams get requirements. How do we know what the code we write should do? How do we obtain enough information to start coding? How do we get the customers to speak with one voice and present their needs clearly? Where do we start on each story? How do we get customers to give us examples? How do we use those to write story tests?

This chapter explains our strategy for creating business-facing tests that support the team as it develops each story. Let's start by talking more about requirements.

THE REQUIREMENTS QUANDARY

Just about every development team we've known, agile or not, struggles with requirements. Teams on traditional waterfall projects might invest months in requirements gathering only to have them be wrong or quickly get out of date. Teams in chaos mode might have no requirements at all, with the programmers making their best guess as to how a feature should work.

Agile development embraces change, but what happens when requirements change during an iteration? We don't want a long requirements-gathering period before we start coding, but how can we be sure we (and our customers) really understand the details of each story?

In agile development, new features usually start out life as stories, or groups of stories, written by the customer team. Story writing is not about figuring out implementation details, although high-level discussions can have an impact on dependencies and how many stories are created. It's helpful if some members of the technical team can participate in story-writing sessions so that they can have input into the functionality stories and help ensure that technical stories are included as part of the backlog. Programmers and testers can also help customers break stories down to appropriate sizes, suggest alternatives that might be more practical to implement, and discuss dependencies between stories.

Stories by themselves don't give much detail about the desired functionality. They're usually just a sentence that expresses who wants the feature, what the feature is, and why they want it. "As an Internet shopper, I need a way to delete items from my shopping cart so I don't have to buy unwanted items" leaves a

lot to the imagination. Stories are only intended as a starting point for an on-going dialogue between business experts and the development team. If team members understand what problem the customer is trying to solve, they can suggest alternatives that might be simpler to use and implement.

In this dialogue between customers and developers, agile teams expand on stories until they have enough information to write appropriate code. Testers help elicit examples and context for each story, and help customers write story tests. These tests guide programmers as they write the code and help the team know when it has met the customers' conditions of satisfaction. If your team has use cases, they can help to supplement the example or coaching test to clarify the needed functionality (see Figure 8-3).

In agile development, we accept that we'll never understand all of the requirements for a story ahead of time. After the code that makes the story tests pass is completed, we still need to do more testing to better understand the requirements and how the features should work.

After customers have a chance to see what the team is delivering, they might have different ideas about how they want it to work. Often customers have a vague idea of what they want and a hard time defining exactly what that is. The team works with the customer or customer proxy for an iteration and might deliver just a kernel of a solution. The team keeps refining the functionality over multiple iterations until it has defined and delivered the feature.

Being able to iterate is one reason agile development advocates small releases and developing one small chunk at a time. If our customer is unhappy with the behavior of the code we deliver in this iteration, we can quickly rectify that in the next, if they deem it important. Requirements changes are pretty much inevitable.

We must learn as much as we can about our customers' wants and needs. If our end users work in our location, or it's feasible to travel to theirs, we should sit with them, work alongside them, and be able to do their jobs if we can. Not only will we understand their requirements better but we might even identify requirements they didn't think to state.

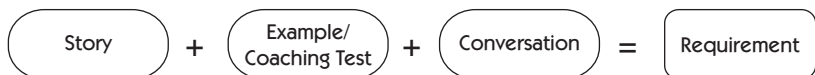


Figure 8-3 The makeup of a requirement

Tests need to include more than the customers' stated requirements. We need to test for post conditions, impact on the system as a whole, and integration with other systems. We identify risks and mitigate those with tests as needed. All of these factors guide our coding.

Common Language

We can also use our tests to provide a common language that's understood by both the development team and the business experts. As Brian Marick [2008] points out, a shared language helps the business people envision the features they want. It helps the programmers craft well-designed code that's easy to extend. Real-life examples of desired and undesired behavior can be expressed so that they're understood by both the business and technical sides. Pictures, flow diagrams, spreadsheets, and prototypes are accessible to people with different backgrounds and viewpoints. We can use these tools to find examples and then easily turn those examples into tests. The tests need to be written in a way that's comprehensible to a business user reading them yet still executable by the technical team.

Business-facing tests also help define scope, so that everyone knows what is part of the story and what isn't. Many of the test frameworks now allow teams to create a domain language and define tests using that language. Fit (Functional for Integrated Framework) is one of those.

■ More on Fit in Chapter 9, "Toolkit for Business-Facing Tests that Support the Team."

The Perfect Customer

Andy Pols allowed us to reprint this story from his blog [Pols, 2008]. In it, he shows how his customer demanded a test, wrote it, and realized the story was out of scope.

On a recent project, our customer got so enthusiastic about our Fit tests that he got extremely upset when I implemented a story without a Fit test. He refused to let the system go live until we had the Fit test in place.

The story in question was very technical and involved sending a particular XML message to an external system. We just could not work out what a Fit test would look like for this type of requirement. Placing the expected XML message, with all its gory detail, in the Fit test would not have been helpful because this is a technical artifact and of no interest to the business. We could not work out what to do. The customer was not around to discuss this, so I just went ahead and implemented the story (very naughty!).

What the customer wanted was to be sure that we were sending the correct product information in the XML message. To resolve the issue, I suggested that we have a Fit test that shows how the product attributes get mapped onto the XML message using XPath, although I still thought this was too technical for a business user.

We gave the customer a couple of links to explain what XPath was so that he could explore whether this was a good solution for him. To my amazement, he was delighted with XPath (I now know who to turn to when I have a problem with XPath) and filled in the Fit test.

The interesting bit for me is that as soon as he knew what the message looked like and how it was structured, he realized that it did not really support the business—we were sending information that was outside our scope of our work and that should have been supplied by another system. He was also skeptical about the speed at which the external team could add new products due to the complex nature of the XML.

Most agile people we tell this story to think we have the “perfect customer!”

Even if your customers aren't perfect, involving them in writing customer tests gives them a chance to identify functionality that's outside the scope of the story. We try to write customer tests that customers can read and comprehend. Sometimes we set the bar too low. Collaborate with your customers to find a tool and format for writing tests that works for both the customer and development teams.

It's fine to say that our customers will provide to us the examples that we need to have in order for us to understand the value that each story should deliver. But what if they don't know how to explain what they want? In the next section, we'll suggest ways to help customers define their conditions of satisfaction.

Eliciting Requirements

If you've ever been a customer requesting a particular software feature, you know how hard it is to articulate exactly what you want. Often, you don't really know exactly what you want until you can see, feel, touch and use it. We have lots of ways to help our customers get clarity about what they want.

Ask Questions

Start by asking questions. Testers can be especially good at asking a variety of questions because they are conscious of the big picture, the business-facing