The Addison-Wesley Signature Series

A Kent Beck Signature Book

User Stories Applied

For Agile Software Development

Mike Cohn
Foreword by Kent Beck

User Stories Applied

- Domain experts can make excellent user proxies but must avoid the temptation to write stories for a product that only someone with their expertise can use.
- Customers, those who make the purchasing decision, can make great user proxies if in close communication with the users for whom they are purchasing the software. Obviously, a customer who is also a user is a fantastic combination.
- In order to be good user proxies, trainers and technical support personnel must avoid the temptation to focus too narrowly on the aspects of the product they see every day.
- This chapter also looked briefly at some techniques for working with user proxies, including the use of user task forces, using multiple user proxies, competitive analysis, and releasing early to get user feedback.

Developer Responsibilities

- You have the responsibility to help your organization select the appropriate customer for the project.
- You are responsible for understanding how different types of user proxies will think about the system being built and how their backgrounds might influence your interactions.

Customer Responsibilities

- If you will not be a user of the software, you are responsible for knowing which categories of user proxy describe you.
- You are responsible for understanding what biases you may bring to the project and for knowing how to overcome them, whether by relying on others or some other means.

Questions

5.1 What problems can result from using the users' manager as a proxy for the users?

5.2 What problems can result from using a domain expert as a proxy for users?

Chapter 6

Acceptance Testing User Stories

One reason for writing acceptance tests is to express many of the details that result from the conversations between customers and developers. Rather than writing lengthy lists of "The system shall…" style requirements statements, tests are used to fill in the details of a user story.

Testing is best viewed as a two-step process: First, notes about future tests are jotted on the back of story cards. This can be done any time someone thinks of a new test. Second, the test notes are turned into full-fledged tests that are used to demonstrate that the story has been been correctly and fully coded.

As an example of test reminders you may write on the back of a card, the story "A company can pay for a job posting with a credit card" may have the following written on the back of its card:

- Test with Visa, MasterCard and American Express (pass).
- Test with Diner's Club (fail).
- Test with good, bad and missing card ID numbers.
- Test with expired cards.
- Test with different purchase amounts (including one over the card's limit).

These test notes capture assumptions made by the customer. Suppose the customer in the BigMoneyJobs example writes the story "A Job Seeker can view details about a specific job." The customer and a developer discuss the story and identify a set of facts that will be displayed about a job—title, description, location, salary range, how to apply, and so on. However, the customer knows that not all companies will provide all of this information and she expects the site to handle missing data. For example, if no salary information is provided, the customer does not even want the "Salary range" label shown on the screen. This should be reflected as a test because the programmer may *assume* that the

job posting part of the system will require every job posting to include salary information.

Acceptance tests also provide basic criteria that can be used to determine if a story is fully implemented. Having criteria that tell us when something is done is the best way to avoid putting too much, or too little, time and effort into it. For example, when my wife bakes a cake, her acceptance test is to stick a toothpick into it; if the toothpick comes out clean, the cake is done. I acceptance test her cake by running a finger through the frosting and tasting.

Write Tests Before Coding

Acceptance tests provide a great deal of information that the programmers can use in advance of coding the story. For example, consider "Test with different purchase amounts (including one over the card's limit)." If this test is written before a programmer starts coding, it will remind her to handle cases in which a purchase is declined because of insufficient credit. Without seeing that test, some programmers will forget to support this case.

Naturally, in order for programmers to benefit in this way, the acceptance tests for a story must be written before programming begins on that story. Tests are generally written at the following times:

- whenever the customer and developers talk about the story and want to capture explicit details
- as part of a dedicated effort at the start of an iteration but before programming begins
- whenever new tests are discovered during or after the programming of the story

Ideally, as the customer and developers discuss a story they reflect its details as tests. However, at the start of an iteration the customer should go through the stories and write any additional tests she can think of. A good way to do this is to look at each story and ask questions similar to the following:

- What else do the programmers need to know about this story?
- What am I assuming about how this story will be implemented?
- Are there circumstances when this story may behave differently?
- What can go wrong during the story?

Story Card 6.1 shows an example for a real project that was buidling software for a scanning system. The author of this story has clearly stated what she expects to happen (the newly-scanned pages go into a new document, even if a document is currently open in the software). In this case, the expectation was described as part of the story on the front of the card. It could just as easily have been stated as a test on the back of the card. The important thing is that the expectation is reflected somewhere on the card prior to the programmers starting on the story. If that is not done, it is likely the programmers could have coded different behavior, such as inserting the newly-scanned pages into the current document.

> A user can scan pages and insert them into a new document. If a document is already open, then the app should prompt and close the current document.

Story Card 6.1 Conveying expectations to the programmers.

The Customer Specifies the Tests

Because the software is being written to fulfill a vision held by the customer, the acceptance tests need to be specified by the customer. The customer can work with a programmer or tester to actually create the tests, but minimally the customer needs to specify the tests that will be used to know when a story has been correctly developed. Additionally, a development team (especially one with experienced testers on it) will usually augment some of the stories with tests they think of.

Testing Is Part of the Process

I recently worked with a company where the tester got her understanding of the software from the programmers. The programmers would code a new feature, they'd explain it to the tester, and the tester would then validate that the program worked as described. Quite often the program would pass these tests but would then be plagued with errors once users started working with it. The problem, of course, was that the tester was testing that the programmer did

what she said she did. Without involvement from customers or users, no one was testing that the software did what they wanted it to do.

With user stories it is vital that testing be viewed as part of the development process, not something that happens "after coding is done." Specifying tests is often a shared responsibility of a product manager and a tester. The product manager will bring her knowledge of the organizational goals driving the project; the tester will bring his suspicious mindset. At the start of an iteration they will get together and specify as many initial tests as they can think of. But it doesn't stop there, and it doesn't stop with them getting together once a week. As the details of a story are worked out, additional tests are specified.

How Many Tests Are Too Many?

The customer should continue to write tests as long as they add value and clarification to the story. It is probably not necessary to write a test to confirm that charges cannot be placed on an expired Visa card if you've already written such a test for expired MasterCards.

Also, keep in mind that a good programming team will have unit tests in place for many of the low-level cases. For example, the programming team should have unit tests that correctly identify February 30 and June 31 as invalid dates. The customer is not responsible for identifying every possible test. The customer should focus her efforts on writing tests that clarify the intent of the story to the developers.

The Framework for Integrated Test

Acceptance tests are meant to demonstrate that an application is acceptable to the customer who has been responsible for guiding the system's development. This means that the customer should be the one to execute the acceptance tests. Minimally, acceptance tests should be executed at the end of each iteration. Because working code from one iteration may be broken by development in a subsequent iteration, it is important to execute acceptance tests from all prior iterations. This means that executing acceptance tests gets more time consuming with each passing iteration. If possible, the development team should look into automating some or all of the acceptance tests.

One excellent tool for automating acceptance tests is Ward Cunningham's Framework for Integrated Test[1], or FIT for short. Using FIT, tests are written in