Acceptance Test-Driven Development: Better Software through Collaboration

Ken Pugh

ken.pugh@netobjectives.com

Abstract

Defining, understanding, and agreeing on the scope of work to be done is often an area of discomfort for product managers, developers and quality assurance experts alike. Many of the items living happily in our defect tracking systems were born of the difficulty we have in performing said activities. Acceptance testing roots out these defects by reexamining the process we use to define the work to be done and how it is tested. Acceptance test-driven development helps with communication between the business customers, the developers, and the testers. The 5 W's are covered- *what* are acceptance tests, *why* you should use them, *when* they should be created, *who* creates them, and *where* they are used.

Biography

Ken Pugh (ken.pugh@netobjectives.com) is a fellow consultant with Net Objectives (www.netobjectives.com). He helps companies transform into lean-agility through training and coaching. His particular interests are in communication (particularly effectively communicating requirements), delivering business value, and using lean principles to deliver high quality quickly. He also trains, mentors, and testifies on technology topics ranging from object-oriented design to Linux/Unix . He has written several programming books, including the 2006 Jolt Award winner Prefactoring, Interface-Oriented Design, and his latest book Lean-Agile Acceptance Test Driven Development: Better Software Through Collaboration. He has helped clients from London to Boston to Sydney to Beijing to Hyderabad. When not computing, he enjoys snowboarding, windsurfing, biking, and hiking the Appalachian Trail.

Ken has a B.S.E from Duke University and an M.S.E. from University of Maryland

1. *What* Are Acceptance Tests?

Acceptance tests are from the user's point of view – the external view of the system. They examine externally visible effects, such as specifying the correct output of a system given a particular input, as shown in Exhibit 1 Acceptance tests can show how the state of something changes, such as an order that goes from "paid" to "shipped". They also can specify the interactions with interfaces of other systems. In general, they are implementation independent, although automation of them may not be. In addition, acceptance tests can suggest that operations normally hidden from direct testing (such as business rules) should be exposed for testing.

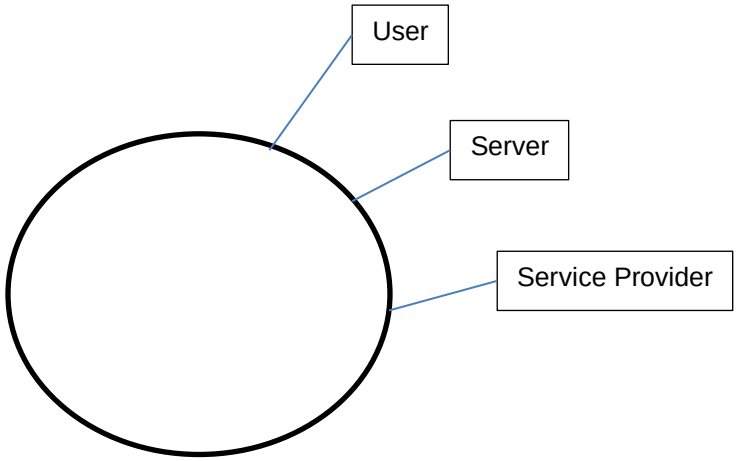

Exhibit 1
What Are Acceptance Tests

2. *Why* Do ATDD?

Teams that follow the acceptance test-driven process have experienced efficiency gains. In one case, rework went down from 60% to 20%. This meant that productivity doubled since the time available for developing new features went from 40% to 80%. In another case, the workflows were working the first time the system was brought up. Getting the business rules right, as in an example we shall see, prevents rework. Because the business customer, developer, and tester are involved in acceptance test creation, there is tighter cross-functional team integration. In addition, passing the acceptance tests visibly demonstrates that the story is complete.

3. *When* Are Acceptance Tests Created?

The value stream map for classical development is shown in Exhibit 2. After eliciting requirements, they are analyzed. A design is created and code developed. Then the system is tested. You can notice many loops go back from test to analysis, design, and coding. These loop backs cause delay and loss of productivity.

Why do these occur? Frequently the cause is misconstructions. In particular, it is misunderstanding the requirements. The loop backs are really feedback to correct these mistakes. There will always be a need for feedback, but quick feedback is better than slow feedback.

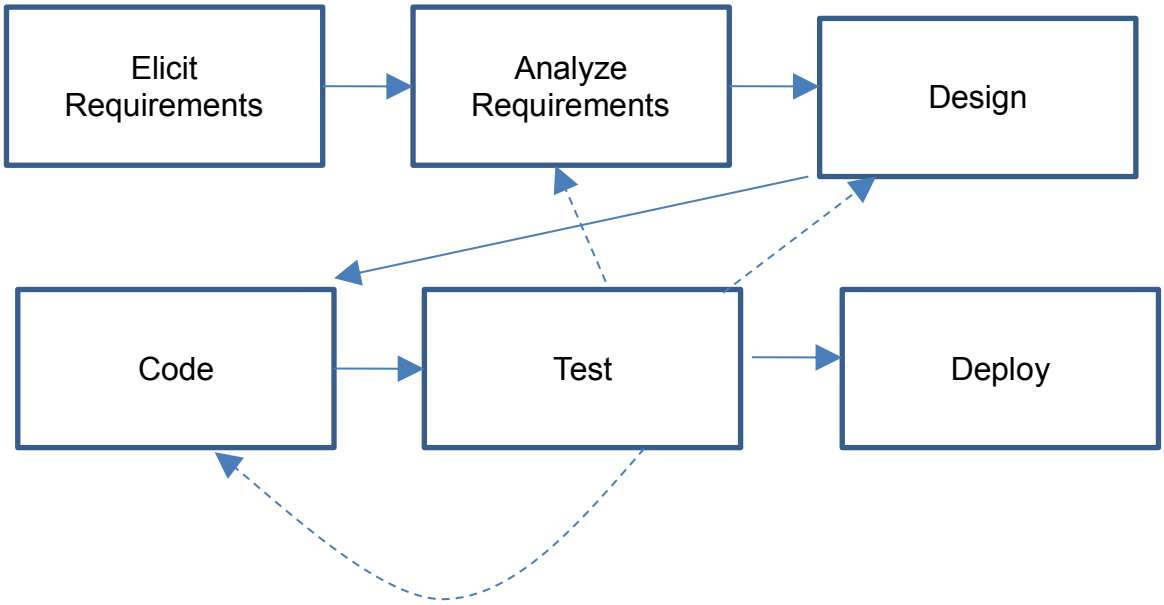

Exhibit 2
Classical Software Development

As you can notice in the revised value stream map in Exhibit 3, the acceptance tests are created when the requirements are analyzed. The developers then code using the acceptance tests. A failing test provides quick feedback that the implementation is not meeting the requirements.

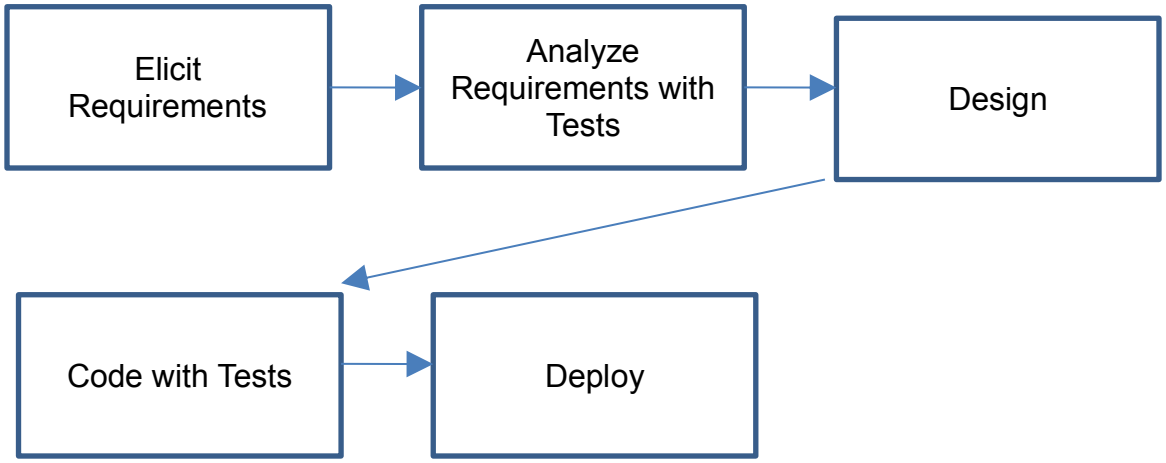

Exhibit 3
Software Development with Acceptance Tests

4. *Who* Authors the Tests?

The tests are authored by the triad – the customer, tester, and developer. At least one example used in the tests should be created by the customer working with the tester and developer. The tester and developer can then create more and have them reviewed by the customer.

The developers connect the test to the system by writing short bits of code. Anyone – customers, developers or testers can run the tests. Acceptance tests can be manual. However, automating acceptance tests allows them to run as regression tests to ensure that new features do not interfere with previously developed features.

Acceptance tests are not a substitute for interactive communication between the members of the triad. However, they provide focus for that communication. The tests are specified in business domain terms. The terms then form a common language that is shared between the customers, developers, and testers.

5. How Do Acceptance Tests Fit Into the Overall Testing Strategy?

Acceptance tests are only a part of the overall testing strategy, as show in Exhibit 4, a diagram adapted from Gerard Meszaros, Mary Poppendieck, and Brian Marick. They are the customer tests that demonstrate the business intent of a system, as shown in the upper left box of the exhibit. The component tests which are beneath them, are technical acceptance tests developed by the architect that help specify the behavior of large modules. Unit tests, the lowest left box, are partially derived from acceptance and component tests, and help the developer to create easy-to-maintain code. The tests on the right – usability, exploratory, and property – examine what are often termed the non-functional aspects. They also need to pass to have a high quality system.

	Per function	*Cross-functional*
Business Facing	Acceptance Tests	Usability Testing
Technology Facing	Component Tests	Exploratory Testing
	Unit Tests	Property Testing

Exhibit 4
Testing Strategies

6. Acceptance Test Example

Suppose you had a requirement which states:

"As the marketing manager, I want to give discounts to repeat customers so that I can increase repeat business."

There are details that go with this story. For example, the customer discount is computed according to a business rule -- Customer Discount Rule. The details of this rule are:

If Customer Rating is Good and the Order Total is less than or equal $10.00,

Then do not give a discount,

Otherwise give a 1% discount.

If Customer Rating is Excellent,

Then give a discount of 1% for any order.

If the Order Total is greater than $50.00,

Then give a discount of 5%.

Now read the rule again and answer one question. For a customer who is Good and has an order of $50.01, what should the discount be?

Depending on how you read the rule, you may come up with one, five, or six percent. The rule is ambiguous. How do we make it clearer? The customer, developer, and tester come up with some examples, which turn into tests.

Suppose they come up with a table of examples, as shown in Exhibit 5. In third set of values, the discount for the case in question should be 1 percent. That's what the business customer wanted. Imagine if the customer had not been consulted and if both the tester and developer had thought it should be 6 percent.

Now these examples are used as acceptance tests. These tests and the requirement are tied together – the tests help clarify the requirement and the requirement forms the context for the tests.

Discount		
Order 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

Exhibit 5
Acceptance Test Examples

7. *Where* Acceptance Tests Are Implemented?

There are at least four ways to implement the tests. They are a testing script, which uses the user interface; a graphical or command line interface; a unit testing framework; and an acceptance testing framework. Let's take a brief look at each.

In the first case, the tester creates a testing script. For example, they logon as a Good customer, start up an order, and put items into it. When the order total is $10.01, they complete the order and make sure

that it shows a $.10 discount. Now they repeat the process for the five other cases, increasing the possibility of carpal tunnel syndrome.

This script needs to be run at least once in order to ensure the discount is computed properly as part of the workflow. However there are three other ways to check for the other cases.

A graphical or command line interface could be created that accessed the module that computed the discount, as shown in Exhibit 5. The tester need only enter the customer rating and order total to determine if the discount is correctly computed.

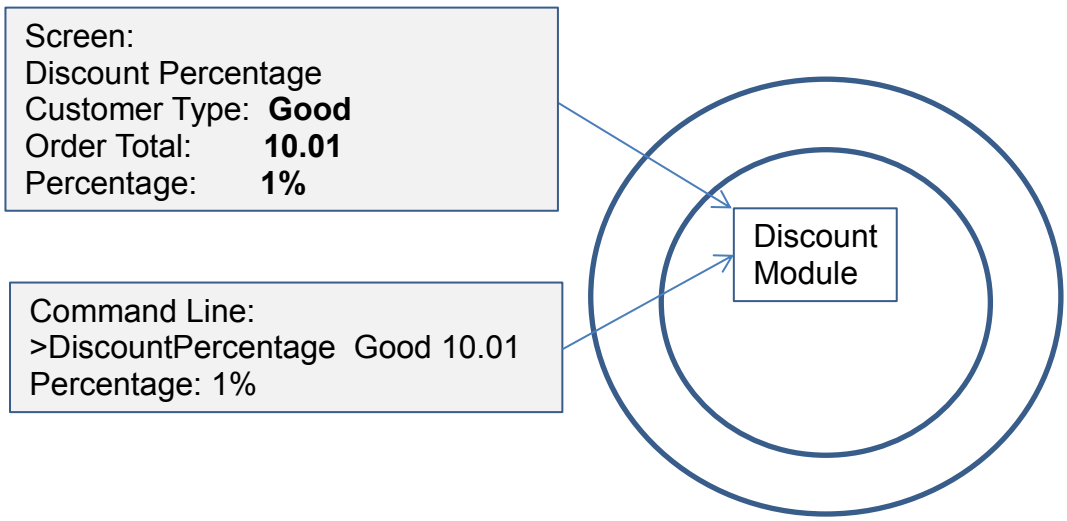

Exhibit 6
Graphical Interface

The developer could create an Xunit test, as shown below. This automates the testing process. However, since the test is in the language of the developer, it can be more difficult to use as a communication vehicle and to ensure that changes in the business rule have been incorporated into the test.

```
class TestCase  {
   testDiscountPercentageForCustomer() {
     SomeClass o = new SomeClass();
     assertEquals(0, o.computeDiscount(10.0, Good));
     assertEquals(1, o.computeDiscount(10.01, Good));
     assertEquals(1, o.computeDiscount(50.01, Good));
     assertEquals(1, o.computeDiscount(.01, Excellent));
     assertEquals(1, o.computeDiscount(50.0, Excellent));
     assertEquals(5, o.computeDiscount(50.01, Excellent));
     }
   }
```

One could use an acceptance test framework, which allows the tests to be readable by the customer. The table shown in Exhibit 7 is from Fit – Framework for Integrated Testing -- but other frameworks, such as Cucumber and Robot Framework have similar tables.

Discount		
Order 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

Exhibit 7
Fit Test

The table becomes a test and is tied to the underlying system through glue code called a fixture. When the test is run, the results appear in the table – green is a pass, red is a fail.

8. Other Uses for Acceptance Tests

In addition to verifying requirements, acceptance tests can be used for other purposes. The number and complexity of the tests can help to estimate the relative effort required to implement a requirement. In the example test, there are six combinations. If there were thirty-six combinations, this would indicate that the effort to realize it would be greater.

The number of acceptance tests that pass relative to the total number of acceptance tests is a relative indicator of how complete is the implementation. In the example, if only one test passed, then the implementation is just started. If all but one test pass, then it is closer to completion.

9. ATDD from the Start

The acceptance test process actually begins at the definition of a feature or capability. For example, the user story about offering discounts is part of marketing initiative. There is a purpose in offering discounts – to increase repeat business. How do you measure the effectiveness of the discount? You need to create an acceptance test, such as "During the next year, 80% of customers will have placed an additional order over their average orders of the past three years." Often acceptance tests as this one are termed project objectives. Objectives should be SMART – specific, measurable, achievable, relevant, and time-boxed.

If this acceptance test passes, then the feature is a success. That is as long as there are no additional business features being added that might affect the outcome, such as providing a personal shopper for every excellent customer. If the acceptance test fails, it may be due to a variety of reasons such as an insufficient discount or a competitor's discount. Or it may be that the objective is not achievable –the economy is such that customers are not buying. In either case, you have a definitive measurement that suggests a course of action such as increasing the discount or abandoning the feature.

10. Review

Acceptance tests represent the detailed requirements for a system. They are developed by the triad--customers, developers, and testers -- as part of requirement definition. They are used by developer and testers during implementation to verify the system. Using acceptance tests can double the efficiency in producing new features.

References

Pugh.K. (2011) Lean-Agile Acceptance Test-Driven Development: Better Software Through Collaboration. Boston: Addison-Wesley