Beginning with the End in Mind: Driving Development with Acceptance Tests

Elisabeth Hendrickson
Quality Tree Software, Inc.
www.qualitytree.com
esh@qualitytree.com

Last updated November 10, 2009

This work is licensed under the Creative Commons Attribution 3.0 United States License. View a copy of this license.
What is ATDD?
Acceptance-Test Driven Development (ATDD) Cycle

(Model developed with Pekka Klärck, Bas Vodde, and Craig Larman.)
ATDD: Discuss

(Slide developed in collaboration with Pekka Klärck.)
ATDD: Develop

(Slide developed in collaboration with Pekka Klärck.)
(Slide developed in collaboration with Pekka Klärck.)
Introducing an Example
Start with the Story

As an administrator, I want users creating accounts to be required to choose secure passwords so that their accounts cannot be hacked by someone using a password guessing program.
Discuss

And if a user provides an insecure password, display an error message.

What does “secure” mean to you?

At least 6 characters with at least one letter, one symbol, and one number.

Who’s in the room?
Product Owner, Testers, Developers, and anyone else who will touch the story.
Capture Concrete Expectations and Examples

<table>
<thead>
<tr>
<th>Password</th>
<th>Valid?</th>
</tr>
</thead>
<tbody>
<tr>
<td>“p@ssw0rd”</td>
<td>Yes</td>
</tr>
<tr>
<td>“p@s5”</td>
<td>No</td>
</tr>
<tr>
<td>“passw0rd”</td>
<td>No</td>
</tr>
<tr>
<td>“p@ssword”</td>
<td>No</td>
</tr>
<tr>
<td>“@#$$1234”</td>
<td>No</td>
</tr>
</tbody>
</table>

Can be expressed as “Given - When - Then”

- **Given** a user is creating an account
- **When** they specify an insecure password
- **Then** they see a message, “Passwords must be at least 6 characters long with at least one letter, one number, and one symbol.”

Or can be expressed in tables

Or in other formats depending on the Framework
Why ATDD?
Reason #1: Drive Out Ambiguity and Clarify Expectations
This is not an Argument about a Bug

"Bug Triage Meeting"
The Tuesday before release.

It's a bug.
No it's not.
Is too.
IS NOT.
IS TOO!
NOT NOT NOT!

Whether or not it's a bug, if we make a change we'll blow the schedule.
Acceptance Tests Define Scope
A Short Digression on ATDD-Friendly Tools
Examples of ATDD-Friendly Frameworks

- Cucumber: a Ruby-based BDD tool that supports “Given-When-Then”
- Fitnesse: a table-driven framework that uses a wiki for displaying and editing tests
- Robot Framework: keyword-driven framework that supports text or tables
- Concordion: Java-based framework for expressing expectations in prose
Frameworks, Interfaces, and Drivers

Created in collaboration. Format defined by the framework.

Code written during development in a programming language determined by the framework.

GUI Driver (e.g. SeleniumRC)

Test "Fixture"

Natural Language Expectations

Implementation

GUI

Public API

"Guts"

Other interfaces
Characteristics of ATDD-Friendly Frameworks

- Support expressing expectations in a language and format that fits the context
- Support collaboration among the whole team including developers, testers, & the product owner
- Connect expectations to the system under test with a minimum of test code ("fixtures," "libraries," "steps") to leverage expectations as executable requirements
- Play nicely with source control systems and continuous integration
- Pluggable to support a variety of interfaces
Contrasting View: Traditional Test Automation

Automated Test Scripts:
Combination of business-facing expectations and implementation details.

Written or recorded after the fact. Expectations are translated, not leveraged.

GUI
Public API
“Guts”
Other interfaces

Implementation
Back to the Example...
Take the Acceptance Tests…

<table>
<thead>
<tr>
<th>Password</th>
<th>Valid?</th>
</tr>
</thead>
<tbody>
<tr>
<td>“p@ssw0rd”</td>
<td>Yes</td>
</tr>
<tr>
<td>“p@s5”</td>
<td>No</td>
</tr>
<tr>
<td>“passw0rd”</td>
<td>No</td>
</tr>
<tr>
<td>“p@ssword”</td>
<td>No</td>
</tr>
<tr>
<td>“@#$%1234”</td>
<td>No</td>
</tr>
</tbody>
</table>

Given a user is creating an account

When they specify an insecure password

Then they see a message, “Passwords must be at least 6 characters long with at least one letter, one number, and one symbol.”

(Note that these expectations are implementation-agnostic and express just the essence of the expectation.)
...and Write the Code

<table>
<thead>
<tr>
<th>Password</th>
<th>Valid?</th>
</tr>
</thead>
<tbody>
<tr>
<td>“p@ssw0rd”</td>
<td>Yes</td>
</tr>
<tr>
<td>“p@s5”</td>
<td>No</td>
</tr>
<tr>
<td>“password”</td>
<td>No</td>
</tr>
<tr>
<td>“p@ssword”</td>
<td>No</td>
</tr>
<tr>
<td>“@#$%1234”</td>
<td>No</td>
</tr>
</tbody>
</table>

Test “Fixture”

GUI
Public API
“Guts”
Other interfaces
Implementation
Why ATDD?
Reason #2: Make progress visible.
Why ATDD?
Reason #3:
Leverage, Efficiency, and Executable Specifications
Traditional Approaches

Requirements Management System

Traceability Matrix

Test Management System

Technical Specifications
Efficiency, Reusability, Maintainability

The tests define the requirements.

Implementation change? Make a localized update to the test fixture.

New interface? Just add a test fixture.

Natural Language Expectations

Test “Fixture”

(No reconciling multiple, duplicate artifacts.)

(Preserve valid expectations.)

(New Interface)

(Leverage relevant expectations.)
How does ATDD fit with the rest of the process?
### ATDD: Part of an Agile Testing Strategy

<table>
<thead>
<tr>
<th></th>
<th>Support</th>
<th>Critique</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Business-facing</strong></td>
<td><strong>Acceptance Tests</strong></td>
<td><strong>Exploratory Testing</strong></td>
</tr>
<tr>
<td><strong>Code-facing</strong></td>
<td><strong>Unit Tests</strong></td>
<td><strong>Reviews, inspections, pairing, code quality metrics</strong></td>
</tr>
</tbody>
</table>

(A variation on Brian Marick’s Agile Testing Quadrants as published in his essay “Agile Testing Directions”)
Tests are Versioned with the Code

- Source Control
  - Natural Language Expectations
  - Test "Fixture" Code
  - Unit Tests
  - Production Code
Tests Execute as Part of the Automated Build

Continuous Integration

Used with permission.
Why ATDD?
Reason #4:
No more bugs. (No, I’m not kidding. But yes, there is a catch.)
Zero Tolerance for Bugs
But Not Everything is a Bug

In this context, a **BUG** is behavior that violates the letter or spirit of the Product Owner’s expectations for the implemented story.

If the behavior does not violate expectations related to the implemented stories, it’s an item for the backlog.
Given all that…
Why **not** ATDD?

