Acceptance Test Driven Development

Mitigating the Risks of Enterprise Software Development

Rod Claar
CSM, CSPO, CSP, CSD, CST

Agile Practice Leader North America
http://rod-claar.net
http://www.Ripple-Rock.com
rod.claar@ripple-rock.com

© 2011 Rod Claar & ripplerock
RippleRock Services

Organisation
- Agile Readiness Assessment
- Stakeholder Workshops
- ALM Process Design
- Organisational Transformation

Team/Project
- Project/Team Selection
- Team Workshops
- Customised Agile Courses
- ALM Process Optimisation

Team/Project
- Certified Scrum Courses
- Agile Coaching >> Mentoring

ALM Tools
- Tools Assessment
- Tools Implementation
- Tools Integration
- Tools Customisation
- Tools Optimisation

Agile Transformation

Effective Scrum Developer Using .NET
With Visual Studio 2010
As a .Net Developer on a Scrum Team I Want to Effectively Write Code that is Correct and Delivers the Required Business Value

What are the big risks in Enterprise Software Development?
What is Enterprise Development Like?

A Minefield With Poisonous Snakes and Man-Eating Tigers!

There Are Lots of Risks!

• Building the Wrong Thing
• Being Late to the Party
• Quality and/or Performance Issues
Building the Wrong Thing

- Communication Issues
  - Stakeholders to Product Owner
  - Product Owner to Team
  - Team (Member) to Team (Member)
- The World Changes

Being Late to the Party

- Requirements Analysis takes too long
- Development takes too long
- Testing takes too long
- Too many features are built
- Poor Code Quality
- Code Base is “Viscous”
- Solution Too Complex / Hard to Add New Stuff
- See #1 (Building the Wrong Thing)
Quality and/or Performance Issues

- Bugs, bugs, bugs!
- User interface problems
- User scenarios take too long to complete
- Application is too complex

Mitigating These Risks

- How can we make communications more clear without doing too much up-front planning?
- How do we guard against the certainty that the world will change?
- How can we shorten the time required for analysis, development and testing?
- How can we narrow the scope of things being developed effectively?
- How can we build quality in?
How?

• Scrum
  – Iterative and Incremental
  – Team Based
  – Empirical, Inspect and Adapt

• A dedication to quality, learning about quality and improving quality
  – Quality can’t be a variable in the project
  – XP Practices

• Acceptance Test-Driven Development
We Do Acceptance Testing Now!

- Well, maybe.
- Automated tests written and failing before the code is complete.
- Automated tests built on specific examples of expected behavior.
- Automated tests failing in plain sight up until they don’t.

Sources of Requirements

- System requirements come from many sources
- Stakeholders often have little practical knowledge of software development
- Quality is often “Done at a reasonable cost”
- The requirements are constantly changing
Requirements Feedback Loops

- Assume we know, write the code and depend on a Scrum feedback to check our understanding.
- Assuming we know, ask a few questions to validate our assumption.
- State the requirements back using different words in a different document.
- Or...

Requirements by Example

- Create some examples of the expected result and validate them with the source of the requirements.
- Convert Celsius to Fahrenheit.
  - Given 0 degrees C, return 32 degrees F
  - Given 100 degrees C, return 212 degrees F

<table>
<thead>
<tr>
<th>celsius to fahrenheit</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Celsius</td>
<td>Convert?</td>
</tr>
<tr>
<td>0</td>
<td>32</td>
</tr>
<tr>
<td>100</td>
<td>212</td>
</tr>
</tbody>
</table>
Acceptance Testing with FitNesse using Slim

Behind the Scenes

© 2011 Rod Claar & ripplerock

A Little History...

• First there was FIT, the Framework for Integrated Testing from Ward Cunningham.
• Then came FitNesse a wiki based test management system that uses FIT to run the tests.
• Then came a lot of other FIT implementations, some better than the original.
• But they were all different.

© 2011 Rod Claar & ripplerock
Enter the Simple List Invocation Method

- Slim is an alternative to FIT.
- It is built into FitNesse.
- Slim keeps the HTML processing, comparisons and results output in FitNesse.
- Slim is very slim. It is easy to port and because all the work is done in FitNesse, the ports will be as nearly identical as the platform allows.

An Example of Slim

Should I Buy Milk?
If I don’t have any cash and we did not pay the credit card bill, even if we don’t have any milk, I can’t go to the store for more. If I have at least $10 and we don’t have any milk, I can go to the store for more. If I don’t have any cash but the credit card is good to go, and we are out of milk, I can go get more. If I have at least $10 and the credit card is good and we are out of milk, I can go get more.

Typical example:  ShouldIBuyMilk

<table>
<thead>
<tr>
<th>Should I Buy Milk</th>
</tr>
</thead>
<tbody>
<tr>
<td>cash in wallet</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>0</td>
</tr>
</tbody>
</table>
When the Test Runs

Should I Buy Milk?
If I don’t have any cash and we did not pay the credit card bill, even if we don’t have any milk, I can’t go to the store for more. If I have at least $10 and we don’t have any milk, I can go to the store for more. If I don’t have any cash but the credit card is good to go, and we are out of milk, I can go get more. If I have at least $10 and the credit card is good and we are out of milk, I can go get more.

Typical example: Slim_example.ShouldIBuyMilk

<table>
<thead>
<tr>
<th>cash in wallet</th>
<th>credit card</th>
<th>pints of milk remaining</th>
<th>go to store?</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>no</td>
<td>0</td>
<td>no</td>
</tr>
<tr>
<td>10</td>
<td>no</td>
<td>0</td>
<td>yes</td>
</tr>
<tr>
<td>0</td>
<td>yes</td>
<td>0</td>
<td>yes</td>
</tr>
<tr>
<td>10</td>
<td>yes</td>
<td>0</td>
<td>yes</td>
</tr>
<tr>
<td>0</td>
<td>no</td>
<td>1</td>
<td>no</td>
</tr>
</tbody>
</table>

Requirements by Example

- Tables record business process conversation decisions
  - Names
  - Rules
- Fixture code connects the examples to the implementation
- Results and Reports measure progress
Test Results

- Identical to the Slim tables except
  - Output cells
    - Green if fixture returns expected result (pass)
    - Red if fixture returns any other result (fail)
      - Expected value
      - Actual value
    - Yellow if the fixture cannot process the inputs or outputs (error)
    - Gray if the test does not specify an expected value outputs the result (pass)

Table Types

- Decision Table
  - Supplies the inputs and outputs for decisions.
- Query Table
  - Supplies the expected results of a query.
- Script Table
  - A series of actions and checks.
- Scenario Table
  - A table that can be called from other tables.
- Table Table
  - Whatever you want it to be!
- Import
  - Add a path to the fixture search path.
Timeline for Acceptance Tests

- Product Owner refines stories and acceptance tests from Release Planning meeting, a few days before the Iteration Planning meeting
- Developers/Testers add more detail tests in the Iteration Planning meeting
- Developers/Testers continue to flesh out in the Iteration – failing tests until code is implemented
- Developers get tests to pass
- Becomes part of the Regression test suite when story is accepted

Acceptance Testing with Slim

Behind the Scenes
A Demonstration Project - Professor Funk and the FunkMasterM2010

Making Fitness Tests Pass
The Payoff!