Proving a Problem is Solved

A developers perspective on requirements testing.
INTRODUCTION

Your presenter
A quick overview
A Little About Me

Carl Wilson
Software Configuration Manager
Open Planets Foundation

Email : carl@openplanetsfoundation.org
Skype : carl.f.wilson
GitHub : carlwilson
Twitter : @openplanets
Google+ : carl@openplanetsfoundation.org
What I Do.....

• The Open Planets Foundation technical dept.
• OPF Events
• OPF Project work
  ▪ SPRUCE
  ▪ SCAPE
• My main goal is to encourage and facilitate community development of high quality digital preservation tools.
Overview

• Defining Requirements?
  ▪ Specifying software systems.
  ▪ What makes a good requirement?

• Software Development Practices
  ▪ Who’d win a fight? Agile vs. Waterfall methodologies.
  ▪ Thinking testability at every step.
  ▪ Open communication and simplicity.

• Thought into Action?
  ▪ Tools and practices to test requirements.
Specifying software systems.
Requirements, what are they good for?
Knowing when you’re done AKA testing your requirements.

DEFINING REQUIREMENTS
Why Specify Requirements?

• The Bottom Line
  Requirements are the contract between the user and the developer.

• When Procuring a Solution
  Requirements provide some of the fine details of the contract between procurer and supplier.

• In Theory…..
  - The customer knows they got what they wanted.
  - The supplier knows when they’ve delivered.
  - We get nice reporting metrics as the project progresses.
The 9 Virtues of Requirements

- So Wikipedia says, edited highlights ;) :

<table>
<thead>
<tr>
<th>Virtue</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unitary (Cohesive)</td>
<td>The requirement addresses one and only one thing.</td>
</tr>
<tr>
<td>Complete</td>
<td>The requirement is fully stated in one place.</td>
</tr>
<tr>
<td>Consistent</td>
<td>The requirement does not contradict any other requirement.</td>
</tr>
<tr>
<td>Non-Conjugated</td>
<td>The requirement is atomic, i.e., it does not contain conjunctions.</td>
</tr>
<tr>
<td>Traceable</td>
<td>The requirement meets all or part of a business need.</td>
</tr>
<tr>
<td>Current</td>
<td>The requirement has not been made obsolete over time.</td>
</tr>
<tr>
<td>Unambiguos</td>
<td>The requirement is concisely stated.</td>
</tr>
<tr>
<td>Specify Importance</td>
<td>The requirement must specify a level of importance.</td>
</tr>
<tr>
<td>Verifiable</td>
<td>The implementation of the requirement can be determined.</td>
</tr>
</tbody>
</table>
Traceable and Verifiable

• I’d like to champion two attributes:
  ▪ Traceable
  ▪ Verifiable

• And the greatest of these is VERIFIABLE

• A truly verifiable requirement isn’t:
  Ambiguous, conjugated (un-atomic), inconsistent (contradicts another test), though static analysis may be required to ensure completeness
Who’d win a fight: Agile vs. Waterfall methodologies?
A few first hand observations on testing and development.
Simplicity, openness and communication.

SOFTWARE DEVELOPMENT PRACTISES
Agile vs. Waterfall Methods

• Not trying to settle the great debate in software development.
• It’s possible to treat methodologies as toolkits.
• The real procurement issues:
  ▪ Specifying what’s to be done.
  ▪ Proving it’s done.
• Between the two lies complexity and miscommunication.
Before I Started in IT....

• My first experience of poorly communicated requirements.

• Who defines when a stone’s large?
  ▪ The supplier (my boss): ≥ a tennis ball
  ▪ The customer: ≥ a golf ball

• My first experience of working evenings and weekends re-picking stones over 8 acres....
Early days in IT

• Organisation in hurry to implement feature.
• The main test developer on leave.
• Feature developer green and keen on golf.
• So just run the dev tests, it’s a minor change.
• Result: back from the golf course early and working late to remove 150,000 duplicate orders from the live system
Coil Plate Mill

Working for British Steel / Corus circa 1999

Scene of my most spectacular real world test failure
Where Waterfall Meets Agile

• Corus a waterfall project over 2 years, BUT:
  ▪ Replacing and enhancing an existing system, one component at a time.
  ▪ Access to business owner, domain experts (metallurgists) in the same office, and end users on site, a two mile car journey away.
  ▪ Open and accessible communication and feedback opportunities.
Real Testable Specifications

• Pension Benchmarking & Attribution

• Requirements Provided by:
  ▪ Financial Analysts
  ▪ Delivered as a set of spreadsheets
  ▪ Reserved another set for testing
  ▪ When software gave the same answers as the spreadsheet, your done

• Client site deployment was another story
What Have I Learned?

• Developing software is the process taking an idea and making it real.
• Clear communication of ideas is a pre-requisite.
• The feedback loop between users, analysts, testers, and developers should be open, honest and regular (think constant).
• Decompose the problem into discrete testable elements.
• Think testability from the ground up.
• Delivering working software shouldn’t be a big deal.
Building testing into the development process.
Connecting developer and acceptance tests.
Automated testing and continuous delivery.

THOUGHT INTO ACTION?
Who’s the Driver?

• Test Driven Development
  Unit Tests : Build the thing right
  ▪ Tools and processes for developers
  ▪ Write a failing test.
  ▪ Write the code to make the test pass, and repeat

• Behaviour Driven Development
  Acceptance Tests : Build the RIGHT THING
  ▪ Tools and processes for teams, based on TDD
  ▪ Define the system in terms of required behaviour
  ▪ Link these specifications to developer tests
Cucumber: A BDD Tool

• Designed specifically to help business stakeholders get involved in writing acceptance tests.

• Provides the sandwich filling between Acceptance Tests and Unit Tests, in a variety of mixable flavours:
  ▪ Integration tests
  ▪ Browser testing
  ▪ Smoke tests
  ▪ And so on....
Cucumber: Encouraging Communication

• Facilitates the discovery and use of a ubiquitous language for project teams.
• Tests written collaboratively by the team, encouraging clear communication.
• Cucumber tests written in a medium and language that business stakeholders understand.
• Cucumber tests interact directly with the code.
Cucumber: Managing Complexity

• Decompose the system into FEATURES, a low level unit of functionality e.g. customer registration

• A feature is made up of TESTABLE scenarios, providing detailed examples of desired behaviour as STEPS:
  - GIVEN some condition
  - WHEN some action / criteria
  - THEN desired result
  - AND further result.....
Cucumber: A Little Detail

• Cucumber test cases are called scenarios, scenarios are made up of steps.
• The business-facing parts of the test suite are grouped into features and stored in feature files.
• Feature file syntax known as Gherkin.
• Below the hood step definitions translate business-facing steps into code.
Cucumber: Testing Stack

Business Facing
- Project
- Features
- Scenarios
- Steps
- Step Definitions
- Support Code
- Automation Library
- System

Technology Facing
Putting it all together

• Continuous Integration
  ▪ Automated build and testing of project
  ▪ Ideally at every code change
  ▪ Can run any kind of automated test
  ▪ Up to date results should always be visible to the whole team.

• Continuous Delivery
  ▪ Delivering a working system as BAU
  ▪ Start with a test system
  ▪ It’s possible to deploy live quickly and often
Footnote: Testing Creatively

• Good testing is NOT easy.
• Adding automated tests to existing code is challenging, refactoring without tests to ensure nothing’s broken.
• Think creatively, black box testing is a good place to start with existing codebases.
• Think creatively, Wizard of Oz testing.....
Final Thoughts

• A bias towards Agile as it encourages:
  ▪ Communication
  ▪ Rapid Feedback

• Specifying systems to a truly testable level of detail is HARD.

• But if YOU, the customer, don’t know how to verify you’ve received what you asked for then you’re almost certain to miscommunicate the idea.
Licensing

This work by Open Planets Foundation is licensed under a Creative Commons Attribution 3.0 Unported License.