Making Test Automation Work in Agile Projects

StarEast 2011

Lisa Crispin

*With Material from Janet Gregory*
Introductions: Experience, Goals
Introduction - Me

- Programming background
- Test automation from mid-90s
- Agile from 2000
  - Many new automation possibilities!
Introduction - You

- Main role on team?
- Programming, automation experience?
- Using agile approach?
- Current level of automation? (Test, CI, deployment, IDEs, SCCS...)

Copyright 2010: Lisa Crispin
Takeaways

Foundation for successful test automation
- “Whole Team” approach
- When to automate
- Apply agile principles, practices
- Good test design principles
- Identifying, overcoming barriers
- Choosing, implementing tools
- First steps

*We won’t do any hands-on automation, but will demo some examples*
Exercise: Your Learning Goals

1. Knowledge
   - Test Methodology
   - Domain Knowledge
   - Requirements Translation

2. Skills
   - Trend in Technology
   - Automation Tools
   - Communication Skills
   - Team Mentality
   - Conflict Resolution

3. People Skills
   - Creative Problem Solving
   - Keeps Up with Changing Environment
   - Creates Alternate Solutions to Problems
   - To Stay Competitive
   - Makes Work Easier
   - To Stay Motivated
   - Constant Improvement

4. Education
   - Learning Stays Relevant
   - tidak diberikan
Why Automate?

- Free up time for most important work
- Repeatable
- Safety net
- Quick feedback
- Help drive coding
- Tests provide documentation
Barriers to Test Automation

What’s holding you back?
Pain and Fear

- Programmers don’t feel manual test pain
- Testers treated as safety net
- Fear
  - Programmers lack testing skills
  - Testers lack programming skills
Initial Investment

- Hump of pain
- Legacy code, changing code
- Tools, infrastructure, time
It’s Worth It

- ROI – explain to management
- “Present value” of automated tests
- Acknowledge hump of pain
Economics of Test Design

- Poor test practices/design = poor ROI
  - Tests had to understand, maintain
- Good test practices/design = good ROI
  - Simple, well-designed, refactored tests
Exercise
Questions?
Getting Over the Hump

- The test automation pyramid
- The agile testing quadrants
- What should be automated
- What shouldn't
- Difficult areas
- Test design
Test Automation Pyramid

- Manual Tests
- GUI Tests
- Acceptance Tests (API Layer)
- Unit Tests / Component Tests
Agile Testing Quadrants

- **Q1**: Technology Facing
  - Unit Tests
  - Component Tests

- **Q2**: Supporting the Team
  - Functional Tests
  - Examples
  - Story Tests
  - Prototypes
  - Simulations

- **Q3**: Business Facing
  - Exploratory Testing
  - Scenarios
  - Usability Testing
  - UAT (User Acceptance Testing)
  - Alpha / Beta

- **Q4**: Critique Product
  - Performance & Load Testing
  - Security Testing
  - “ility” Testing

*Automated & Manual* vs. *Manual*
*Automated* vs. *Tools*
What Should We Automate?

- Quadrant 1 tests
  - Unit, component, TDD
- Quadrant 2 tests
  - Behind GUI, API, web services
- Quadrant 4 tests
  - Load, performance, stress
- Quadrant 3 tests?
  - Leverage automation where useful
What Shouldn’t We Automate?

- Quadrant 2 tests
  - Wizard of Oz, prototyping
- Quadrant 3 tests
  - Usability, UAT, ET
- Tests that will never fail?
  - Assess risk
- ROI not enough
  - One-off tests
Where Should We Be Careful?

- **GUI tests**
  - Need to test the GUI
  - Watch ROI
- **End-to-End tests**
  - Push testing down to lowest level
  - Robust lower-level tests = better ROI
- Remember the Pyramid
Hard to Automate?

- Legacy code
  - Hard to automate, or just lack of skill?
  - “Working Effectively with Legacy Code” – Feathers
  - “Strangling” – Fowler, Thomas
Exercise: Low-Hanging Fruit
Agile Automation Strategy

- What hurts the most
- Layered approach
- Applying agile principles
  - Whole team approach
  - Small chunks/thin slices
  - Smart test design
- Choosing the right tools
What Hurts the Most

- Keep an impediment backlog
- What’s biggest obstacle?
  - Time
  - Tools
  - Code design
  - ...
Multi-Layered Approach

Example:
- Developers address unit tests
- While testers write GUI smoke tests
Whole-Team Approach

- Team responsible for testing, quality
  - Testable architecture
- Team responsible for testing activities
- Whole team has all the skills needed
  - Good code design skills essential
- Team designs for ease of test automation
Exercise: Skills
Simplicity

- Address one or two needs at a time
- Understand the problem first
- Try simplest approach first
- Work in small chunks, thin slices
- Incremental & iterative
Automate a Slice at a Time

Example: 4-step UI to validate, upload profit sharing contribution data

• Thread 1: All four pages with navigation
• Thread 2: Select year, enter description on page 1, display on page 2, browse and upload file on page 2
• Thread 3: Validate data in file, display on page 3
• Thread 4: Persist data, display ‘success’ message on page 4
Thin Slice Example

1. **STEP 2**
   - New Step 3: "Empty"
   - Shows IF System Prop > True

2. **STEP 2**
   - New Step 3: 
     - IF Port. Mgr. is Present
   - Works as it does today
   - IF No Port. MGR Present

3. **STEP 2**
   - New Step 3
   - Fund Selection
   - Uses Port. Mgr.'s portfolio to show pre-selected funds

4. **STEP 2**
   - New Step 3: ALWAYS EDIT
   - Fund Selection
   - Docs
     - After Docs accepted, New Step 3 is read only
Mind Map Example

Delete items from shopping cart

Retrieve?

New order total
Display new cart without deleted item

Display items in the cart

What happens to deleted items?
Save for later?  Wishlist?  Just go away?

Some way to delete
Checkbox?  Quantity?  Delete button or link?

Copyright 2010: Lisa Crispin
Good Test Design

- Essence of each test is clear
  - Readable by business experts
  - Hide incidental details
- Create & use a standard template
- DRY – don’t repeat yourself
  - Extract duplication using macros, modules, variables, classes, mixins
- Pair, peer review
- Use retrospectives, address pain points
  - Refactor
Demo

- Let’s illustrate some test design principles with a simple test in FitNesse
- Your handout has a separate example in Robot Framework.
Iterative Feedback

- Commit to trying new tool/framework/technique for N iterations
- Plan automation tasks for each iteration
- Use retrospective to evaluate
Learn by Doing

- Courage – don’t be afraid to fail
- Use agile coding practices for automation
  - Simple design
  - Pairing
  - Refactoring
  - Object-oriented, libraries, modules
  - Test-first if scripts have logic
- Remember small chunks
- Experiment
Questions About Automation Strategy?
Exercise: Thin Slices

Given this story:

As an Internet shopper, I want to know the shipping cost of an item during checkout based on the shipping address, weight and method.

Assumptions: User has already entered valid shipping address. User will be able to choose different options for different items. The options are USPS, Ground, 2 day and Overnight. PO Boxes are USPS only. Items > 20 lbs are Ground only. API to cost calculator available, takes postal code and weight

Mind map this on a big sheet of paper
Identify a basic end-to-end slice of functionality that can be coded, tested and automated
Choosing Tools

- Must be team decision
- Find time for evaluating
  - Story for tool evaluation
  - Iteration for development team
- Determine requirements
- Focus on goals, problems, not tools.
- Experiment
Understand the Purpose

- What’s being automated?
  - eg. Ajax, SSL support; load; embedded s/w
  - Speeding up exploratory testing, test data
  - Documentation
- Existing tools, environment
  - eg., integration with build process
  - Reporting needs
- Who’s writing, maintaining the tests?
- Who’s using the tests, for what?
What Fits Your Situation

- Existing skills on team
- Language of application under test
- Collaboration needs
- Utilities (automating tedious tasks) vs. Testing
- Life span, future use of tests
Vendor Tools - Pros

- Existing expertise
- Some built on open-source libraries
- Fast ramp-up for non-programmers
- Perceived as safe choice
- Training, support
- Part of existing tool set
- May have robust features
Vendor Tools - Cons

- Tend to be heavyweight
- Tend to be programmer-unfriendly
- Scripts may be brittle, high-maintenance
  - Capture-playback problematic
  - Not designed for long-term maintainability
- Can be pricey
Open-Source Tools - Pros

- Designed by test-infected programmers
- Designed for agile environments
- Designed for maintainability
- Programmer-friendly
- May have excellent support, tutorials, doc
- Easily customized
- Low up-front cost
Open-Source Tools - Cons

- May be difficult for non-programmers
  - Depends on the tool/framework
- Future enhancements may be uncertain
- Training, support can be an issue
- Be sure to look for active development community
Home-Brewed - Pros

- Programmer-friendly
  - Integration with app, IDEs
- Development framework may support
  - Rails, Ruby, Groovy
- Can build on top of existing framework
  - Fit, Slim, Watir, RSpec
- Specifically tailored to needs
- Someone’s there to address problems
Home-Brewed - Cons

- Team needs enough bandwidth, expertise
  - Reporting framework
  - Allow test specification by non-programmers
- Could be harder sell to management
Where To Find Tools

- www.softwareqatest.com/qattls1.html
- www.testingfaqs.org
- www.opensourcetesting.org
- awta.wikispaces.com/2009ToolsList
- groups.yahoo.com/group/agile-testing
Example: My Team’s Tool Choices

- IntelliJ Idea, Eclipse for IDEs
- CruiseControl, Hudson for CI
- JUnit for TDD, unit, component
- FitNesse for functional, behind GUI
- Canoo WebTest for GUI regression smoke tests
- Watir to aid exploratory testing
- JMeter for load, performance testing
- Perl scripts for comparing files, making files human-readable
- Java programs for concatenating forms, cover letters
Exercise: Tools

Browse EE UI Test Run Alone

- Time (in Milliseconds)
- # of Users

Average
Max
Making Test Automation Work

- Time to do it right
- Learning culture
- Testable architecture
- Test data
- Managing tests
Time To Do It Right

- Limit scope, don’t over-commit
- Write automation task cards
- Quality must be team goal
- Long-term, will let you go faster
Learning Culture

- OK to make mistakes
- Lots of small experiments
- Slack
- Evolve right design
Testable Architecture

- Layered architecture
  - eg. UI, business logic, data access
- Ports and Adapters pattern
  - App can work without UI or database
  - Ports accept outside events
  - Adapters convert for human or automated users
Test Data

- Avoid database access when possible
- Setup/Teardown
  - Independent, rerunnable tests
- Canonical data
  - Refresh before each test run
- Customizable data for ET
- Production-like data
  - Get customers to provide example data
Managing Automated Tests

- Documenting tests
  - Tests as documentation
  - Finding what you need
- Running tests – Continuous Integration
- Reporting results
  - Analyze failures
- Test coverage
Tests as Documentation

- Automated tests can be readable by everyone
  - Examples can be turned into tests easily by a framework
  - Automation can be hidden or built in
  - Given/When/Then BDD style is one way
  - “Do” or “Scenario” fixture is another way
- Must work for YOUR company
- Tests automated in CI must pass = documentation is up to date!
Any Example Can Become a Test

<table>
<thead>
<tr>
<th>Firm Type</th>
<th>Entity</th>
<th>Individuals Registration List/Entry</th>
<th>Fiduciary Advisor Option</th>
<th>Fee Pmt</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>RIA</td>
<td>Firm (many to one)</td>
<td>RIA Picklist</td>
<td>Yes if product allows</td>
<td>paid to Entity</td>
<td>Firm, Individual</td>
</tr>
<tr>
<td>B/D</td>
<td>Firm</td>
<td>B/D Picklist</td>
<td>Same</td>
<td>paid to Entity</td>
<td>Firm only</td>
</tr>
</tbody>
</table>

Other:
- No B/D = NA
- No Fiduciary Option
- No B/D on list
- No Fiduciary
- B/D on B/D list
- All B/D on list

- Not Fiduciary
- Not B/D on list
- NA or B/D

- No B/D on list
- NA or B/D

- Selling Agreement

Single Fee Schedule
Given/Then/When Example

Scenario: Valid name search returns results

GIVEN that Kant is a supervisor with employees AND Kant has an employee named Smith WHEN Kant navigates to the employee name search page AND enters the value “S” THEN Kant will see a search result that includes Smith
1. Take out a loan
2. Check the calculated loan payment
3. Post the payment, then receive it
4. Settle and confirm the payment
5. Check the interest, principal, loan balance and default state

### FitNesse “Do” Fixture

<table>
<thead>
<tr>
<th>Loan Processing Fixture</th>
</tr>
</thead>
<tbody>
<tr>
<td>take loan in the amount of 1000</td>
</tr>
<tr>
<td>check</td>
</tr>
</tbody>
</table>
Test Management Tools

- Tool may include management framework
  - FitNesse
    - Wiki – documentation + executable tests
    - Hierarchy
  - Rasta – spreadsheets to organize tests
  - Twist – uses Eclipse IDE

- What problem are you trying to solve?
- Simplest approach
- Check tests into same sccs as production code
Exercise: Tests as Documentation

<table>
<thead>
<tr>
<th>Build Employees Fixture</th>
</tr>
</thead>
<tbody>
<tr>
<td>userId</td>
</tr>
<tr>
<td>1001</td>
</tr>
<tr>
<td>1002</td>
</tr>
<tr>
<td>1003</td>
</tr>
<tr>
<td>1004</td>
</tr>
<tr>
<td>1005</td>
</tr>
<tr>
<td>1006</td>
</tr>
<tr>
<td>1007</td>
</tr>
<tr>
<td>1008</td>
</tr>
</tbody>
</table>

**Operate on Input by Running ADP Test**

Operate Adp Test Fixture
- Operate!
- true

**Make Assertions about ADP Test Results**

<table>
<thead>
<tr>
<th>Check Employee Fixture</th>
</tr>
</thead>
<tbody>
<tr>
<td>userId</td>
</tr>
<tr>
<td>1001</td>
</tr>
<tr>
<td>1002</td>
</tr>
<tr>
<td>1003</td>
</tr>
<tr>
<td>1004</td>
</tr>
<tr>
<td>1005</td>
</tr>
<tr>
<td>1006</td>
</tr>
<tr>
<td>1007</td>
</tr>
<tr>
<td>1008</td>
</tr>
</tbody>
</table>

Copyright 2010: Lisa Crispin
Key Success Factors

- Whole team approach
- Simple approach
- Iterative feedback
- Good test design
- Agile principles, practices
- Learning culture
Succeeding with Test Automation

- Don’t overcommit – budget time to automate tests
  - Well-designed tests mean speed later
- Need testable architecture, design
  - But don’t get stuck, find a way
- Keep feedback loop short – use CI
- Team designs how much coverage is enough
  - Focus on feature coverage, not % of code
- Tests *are* documentation
Exercise: Breaking Barriers
Remember

- It’s a team problem!
- Build foundation of core agile practices
- Design investment, refactoring pay off
- Experiment
- Baby steps
Questions? “Aha” Moments?
Agile Testing: A Practical Guide for Testers and Agile Teams

By Lisa Crispin and Janet Gregory

www.agiletester.ca
Now Available

Beautiful Testing: Leading Professionals Reveal How They Improve Software

Edited by Tim Riley, Adam Goucher

Includes chapter by yours truly
Test Patterns

_Xunit Test Patterns: Refactoring Test Code_

By Gerard Meszaros
Bridging the Communication Gap

Specification By Example and Acceptance Testing

Gojko Adzic
Specification by Example

How successful teams deliver the right software

Gojko Adzic

Case studies from > 50 teams
Agile Test Automation Resources

- dhemery.com/pdf/writing_maintainable_automated_acceptance_tests.pdf
- lisacrispin.com
- janetgregory.ca
- gokjo.net
- exampler.com
- agile-testing@yahoogroups.com
- testobsessed.com
- testingreflections.com
- pairwith.us