Agile Test Automation Strategy

For Anyone and Everyone!

Gerard Meszaros
Agile2012ATAS@gerardm.com
My Background

80's
• Software developer
• Development manager
• Project Manager
• Software architect

90's
• OOA/OOD Mentor
• Requirements (Use Case) Mentor
• XP/TDD Mentor

00's
• Agile PM Mentor
• Test Automation Consultant & Trainer
• Lean/Agile Coach/Consultant

Embedded Telecom

I.T.

Product & I.T.

Much Ado About Agile 2011

Copyright 2011 Gerard Meszaros
Agenda

- **Motivation**
  - The Agile Test Problem
  - The Fragile Test Problem

- Approaches to Test Automation

- Test Automation Strategy

<table>
<thead>
<tr>
<th>Topic</th>
<th>Time</th>
<th>Slides</th>
<th>Start</th>
<th>End</th>
</tr>
</thead>
<tbody>
<tr>
<td>Motivation</td>
<td>11.2</td>
<td>8</td>
<td>2</td>
<td>9</td>
</tr>
<tr>
<td>Exercise 1 - Automation Motivation</td>
<td>10</td>
<td>1</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>Intro to Automation</td>
<td>7</td>
<td>5</td>
<td>11</td>
<td>15</td>
</tr>
<tr>
<td>Exercise 2 - Why not Record &amp; Playback?</td>
<td>10</td>
<td>1</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>Why Automated Tests are Fragile</td>
<td>8.4</td>
<td>6</td>
<td>17</td>
<td>22</td>
</tr>
<tr>
<td>How Agile Automation Changes Things</td>
<td>9.8</td>
<td>7</td>
<td>24</td>
<td>30</td>
</tr>
<tr>
<td>Intro to Example-Driven Development</td>
<td>7</td>
<td>5</td>
<td>32</td>
<td>36</td>
</tr>
<tr>
<td>Managing Scope vs Detail in Examples</td>
<td>15.4</td>
<td>11</td>
<td>38</td>
<td>48</td>
</tr>
<tr>
<td>How to specify workflows</td>
<td>8.4</td>
<td>6</td>
<td>50</td>
<td>55</td>
</tr>
<tr>
<td>Exercise 3 - Workflow Tests (Keyword-Driven)</td>
<td>15</td>
<td>1</td>
<td>56</td>
<td>56</td>
</tr>
<tr>
<td>Using Data-Driven Tests to specify business rules</td>
<td>8.4</td>
<td>6</td>
<td>55</td>
<td>60</td>
</tr>
<tr>
<td>Exercise 4 - Business Rules Test (Data-Driven)</td>
<td>15</td>
<td>1</td>
<td>61</td>
<td>61</td>
</tr>
<tr>
<td>How Tests Interact With the SUT</td>
<td>7</td>
<td>5</td>
<td>62</td>
<td>66</td>
</tr>
<tr>
<td>Test-Driven Architecture</td>
<td>5.6</td>
<td>4</td>
<td>67</td>
<td>70</td>
</tr>
<tr>
<td>Legacy Systems (if time permits)</td>
<td>19.6</td>
<td>14</td>
<td>71</td>
<td>84</td>
</tr>
<tr>
<td>The Role of Unit Tests</td>
<td>8.4</td>
<td>6</td>
<td>85</td>
<td>90</td>
</tr>
<tr>
<td>Test Automation Strategy</td>
<td>14</td>
<td>10</td>
<td>91</td>
<td>100</td>
</tr>
</tbody>
</table>
Product Owner Goal

• Goal: Maximize business value received

Quality is Assumed; Not Managed
Why Quality Often Sucks

• Iron Triangle of Software Engineering:

  Resources

  Quality

  Functionality

   Time

• What about Quality?

  You can fix any three; the fourth is the outcome
Why Quality Often Sucks

• Iron Triangle of Software Engineering:

  Resources  
  Time  
  Quality  
  Functionality

  In Agile, we “Pin” quality using automated tests

• What about Quality?
  You can fix any three; the fourth is the outcome
Speaking of Quality, would you ...

... ask your doctor to reduce the cost of the operation ...

... by skipping the sterile technique?

Test Automation is like hand washing: Improved results but an upfront cost.
Minimizing Cost of Product

Total cost includes:
• developing the software
• verifying the newly built functionality
• verifying old functionality still works
• fixing any bugs found
• Verifying noting was broken by fixes

Agile Test Automation can reduce the cost of all of these activities.
Incremental Development

Initial Concept

Evolved Concept

- Type NF bugs: New Func. is wrong
- Type RB bugs: New bugs in old func. (Regression Bugs)
Exercise 1

• Time to test our little application

• Oh, new build, please retest!

• Another build, please retest!
The Agile Test Problem

Requirements  Development  Testing
The Agile Test Problem

- As development increments reduce in duration, testing needs to be reduced accordingly
The Agile Test Problem

... and traditional approaches to testing no longer work
Anatomy of an Automated Test

<table>
<thead>
<tr>
<th>Test Scenario Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preconditions</td>
</tr>
<tr>
<td>1. Do Something</td>
</tr>
<tr>
<td>2. Check Something</td>
</tr>
</tbody>
</table>

1&2 May be repeated

Clean Up

When ... Then....

Given...

Test Setup

1&2 May be repeated

Test Teardown

Preconditions (State)

Other System

Other System

Other System

Database

Container Services

Business Logic

Interface

Other System

Our System

Adapter
(C)OTS Record&Playback

- User executes tests manually; tool records as tests
- Tool replays tests later without user intervention

The tests are code/data interpreted by the test runner.
Exercise 2

• Record & Playback Test Automation
  – Please record a test against the System Under Test
  – Then, run the test to make sure it works

• New build has been delivered
  – Please run the test against new build
Agenda

• Motivation
  – The Agile Test Problem
  – The Fragile Test Problem

• Changing the Role of Test Automation
• Approaches to Test Automation
• Test Automation Strategy
The Fragile Test Problem

What, when changed, may break our tests accidentally:

– Behavior Sensitivity
  » Business logic
– Interface Sensitivity
  » User or system
– Data Sensitivity
  » Database contents
– Context Sensitivity
  » Other system state

In Agile, these are all changing all the time!
Interface Sensitivity

• **Tests must interact with the SUT through some interface**

• **Any changes to interface may cause tests to fail.**
  
  – **User Interfaces:**
    » Renamed/deleted windows or messages
    » New/renamed/deleted fields
    » New/renamed/deleted data values in lists
  
  – **Machine-Machine Interfaces:**
    » Renamed/deleted functions in API
    » Renamed/deleted messages
    » New/changed/deleted function parameters or message fields

  E.g.: Move tax field to new popup window
Behavior Sensitivity

• Tests must verify the behavior of the system.
  – Behavior also involved in test set up & tear down

• Any changes to business logic may cause tests to fail.
  – New/renamed/deleted states
  – New/changed/removed business rules
  – Changes to business algorithms
  – Additional data requirements

E.g.: Change from GST+PST to HST
Data Sensitivity

• All tests depend on “test data” which are:
  – Preconditions of test
  – Often stored in databases
  – May be in other systems
• Changing the contents of the database may cause tests to fail.
  – Added/changed/deleted records
  – Changed Schema

E.g.: Change customer’s billing terms
Context Sensitivity

- Tests may depend on inputs from another system
  - State stored outside the application being tested
  - Logic which may change independently of our system

- Changing the state of the context may cause tests to fail.
  - State of the container
    » e.g. time/date
  - State of related systems
    » Availability, data contents

E.g.: Run test in a shorter/longer month
Agenda

• **Motivation**

• **Changing the Role of Test Automation**
  – From Defect Detection to Defect Prevention
  – Different Tests for Different Purposes

• **Approaches to Test Automation**

• **Test Automation Strategy**
The Role of Automation in Agile

• **Provide a Safety Net for Change & Innovation**
  – Provide rapid feedback to reduce cost of fixing defects.
    » On demand (Developer) and event driven (CI build)
  – Rapid feedback enables experimentation
    » Don’t have to choose between Quick and Safe

• **Guide Development of the Product**
  – Provide executable examples of what “done” looks like

• **Support Manual Testing**
  – Remove the repetitive drudgery so testers can focus on high value activity by:
    – Automating entire tests, or by
    – automating the steps that can be automated.
How is Agile Test Automation Different?

• **We automate the tests for a different reason**
  – Defect Prevention vs. Detection
  – To communicate requirements
  – To “Pin” the functionality once it’s built

• **We automate the tests a different way**
  – Many different kinds of tests
    » E.g. We don’t rely solely on GUI-based automation
  – Using tools that support collaboration & communication
    » in addition to confirmation

• **We plan the automation based on ROI**
  – Goal isn’t: 100% automation
  – Goal is: To maximize benefit while minimizing cost
Traditional Role of Testing

Critique Product

<table>
<thead>
<tr>
<th>Business Facing</th>
<th>Acceptance Tests</th>
<th>Regression Tests</th>
<th>Usability Tests</th>
<th>Exploratory Tests</th>
</tr>
</thead>
<tbody>
<tr>
<td>Technology Facing</td>
<td>Unit Tests</td>
<td>Component Tests</td>
<td>Property Tests</td>
<td>(Response Time, Security, Scalability)</td>
</tr>
</tbody>
</table>

Report Card

- Functionality: B
- Usability: C
- Scalability: A
- Response: B
- Availability: C

Inspection to find defects is **Waste**

*Shigeo Shingo*

Co-inventor of Toyota Production System

Inspection to prevent defects is **essential**

Much Ado About Agile 2011

Quadrants courtesy of Brian Marrick and Mary Poppendieck

Copyright 2011 Gerard Meszaros
Changing the Role of Testing

Critique Product

Define Product

<table>
<thead>
<tr>
<th>Business Facing</th>
<th>Technology Facing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acceptance Tests</td>
<td>Unit Tests</td>
</tr>
<tr>
<td>Regression Tests</td>
<td>Component Tests</td>
</tr>
<tr>
<td>Usability Tests</td>
<td>Property Tests</td>
</tr>
<tr>
<td>Exploratory Tests</td>
<td>(Response Time, Security, Scalability)</td>
</tr>
</tbody>
</table>

Report Card

<table>
<thead>
<tr>
<th>Functionality</th>
<th>Usability</th>
<th>Scalability</th>
<th>Response</th>
<th>Availability</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>C</td>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
</tbody>
</table>

Prevent anticipatable defects from happening

Find non-anticipatable Defects, ASAP!

Quadrants courtesy of Brian Marrick and Mary Poppendieck
Changing the Role of Testing

Acceptance Tests
Regression Tests
Usability Tests
Exploratory Tests
Property Tests (Response Time, Security, Scalability)

For effective prevention:
1. Tests must be available before development
2. Developers must be able to run tests before check-in
Reducing the Cost to Fix Defects

Cost to understand and fix a defect goes up with the time it takes to discover it.

• Why?

• We can remember where we put the newly inserted defect because
  1. We know what code we were working on
  2. The design of the code is still fresh in our minds

• We may have to change less code
  – Because we wrote less code based on the defect
Continuous Acceptance Testing

- Defines what “Done Looks Like”
  - Several to many tests per User Story / Feature

- Tests executed as soon developer says “It’s Ready”
  - End-of-iteration: OK
  - Mid-iteration: Better

Write StoryTest | Build Code | Test Code | Test Story
                |            |           |
Write StoryTest | Build Code | Test Code | Test Story
Continuous Readiness Assessment!

- Defines what “Done Looks Like”
  - Several to many tests per User Story / Feature

- Executed by developers during development
  - To make sure all cases are implemented
  - To make sure it works before showing to business

- Tests executed as soon developer says “It’s Ready”
  - End-of-iteration: OK
  - Mid-iteration: Better
Prevention: - Building the Right Product

What the customer thought they wanted
What the customer actually asked for
What the customer realized they actually needed

What development thought the customer asked for
What development actually built

What testing thought the customer asked for
What testing actually tested for
Building the Right Product

• How do we eliminate the waste caused by building the wrong product?
  – Missed requirements?
  – Misunderstood requirements?
  – Unneeded functionality?
Building the Right Product

- How do we eliminate the waste caused by building the wrong product?
  - Missed requirements?
  - Misunderstood requirements?
Example-Driven Development

• A.K.A.
  – Acceptance Test Driven Development
  – Behaviour-Driven Development
  – Executable Specification
  – StoryTest-Driven Development

• Concrete examples *flesh* out requirements

• Testers *flush* out missed scenarios...
  ...before development starts
Life Cycle of an Example / Test

User Goal

Feature

Story Title

Story Narrative

Define Acceptance Criteria

Story Scenarios

Make Concrete By Adding Data

Story Examples

Formalization Automation

Executable Examples

Product Development

Satisfied Examples

Much Ado About Agile 2011
Agenda

• Motivation
• Changing the Role of Test Automation
  – From Defect Detection to Defect Prevention
  – Different Tests for Different Purposes
• Approaches to Test Automation
• Test Automation Strategy
Test Automation Pyramid

- Large numbers of very small unit tests
  - Ensures integrity of code
- Smaller number of functional tests for major components
  - Verify integration of units
- Even fewer tests for the entire application & workflow
  - Ensure application(s) support users’ requirements
- Tools to support effective exploratory testing

Pyramid originally proposed by Mike Cohn
Behavior Specification at Right Level

• Specify broad scope at minimum detail
  – E.g. Use least detail when specifying workflow
• Specify most detailed req’ts at narrowest scope
  – E.g. Don’t use workflow when specifying business rules

---

Make examples / tests easy to understand and easy to write
Example: **Mega Bank Requirements**

- Notify user of transactions against their accounts.
- User can configure threshold amount for notification based on any/all of account, transaction type or region, charge category.
- Notification can be sent via e-mail, voice-mail or SMS/IM.
- User can suspend notifications indefinitely or for a defined period of time.
Example: Mega Bank Use Cases

- Configure Notification Threshold
- Suspend Notification
- Resume Notification
- Process Transaction
Example:

**Specifying Notification Workflow**

<table>
<thead>
<tr>
<th>Time now is</th>
<th>9:00AM, 03/18/2008</th>
</tr>
</thead>
<tbody>
<tr>
<td>Customer</td>
<td>bobma</td>
</tr>
<tr>
<td></td>
<td>sets notification threshold to $10,000.00 for all transactions</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Time now is</th>
<th>9:30AM, 03/18/2008</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bank processes</td>
<td>dedit to 10035692877 in the amount of $15,000.00</td>
</tr>
<tr>
<td>Bank processes</td>
<td>debit to 10035692877 in the amount of $9,000.00</td>
</tr>
<tr>
<td>Bank processes</td>
<td>dedit to 10035692877 in the amount of $11,000.00</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>New notifications sent to customer</th>
<th>bobma</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>account</td>
</tr>
<tr>
<td>dedit</td>
<td>10035692877</td>
</tr>
<tr>
<td>dedit</td>
<td>10035692877</td>
</tr>
</tbody>
</table>

**Use Case: Manage Notification Thresholds**

**Use Case: Process Transaction**

**Check output of Use Case: Process Transaction**

*Broad Scope; Minimum Detail; No mention of User Interface!*
Alternate form of Workflow Test:

Given Bobma has account 1003592877
And BobMa sets notification threshold to $10,000 for all transactions
When the bank processes debit for 15,000 to account 1003592877
And the bank processes debit for 9,000 to account 1003592877
And the bank processes debit for 11,000 to account 1003592877
Then bobma receives notification for debit 15,000 to account 1003592877
And bobma receives notification for debit 11,000 to account 1003592877
Specifying Suspension Workflow

Use Case: Manage Notification Thresholds

Use Case: Process Transaction

Use Case: Suspend Notification

Use Case: Resume Notification

Use Case: View Notifications

Example:

<table>
<thead>
<tr>
<th>Time now is</th>
<th>9:00AM, 03/18/2008</th>
</tr>
</thead>
<tbody>
<tr>
<td>Customer</td>
<td>bobma</td>
</tr>
<tr>
<td></td>
<td>sets notification threshold to $10,000.00 for all transactions</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Time now is</th>
<th>9:30AM, 03/18/2008</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bank processes</td>
<td>debit to 10035692877 in the amount of $15,000.00</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Time now is</th>
<th>10:00AM, 06/16/2008</th>
</tr>
</thead>
<tbody>
<tr>
<td>Customer</td>
<td>bobma</td>
</tr>
<tr>
<td></td>
<td>suspends notifications on account 10035692877</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Time now is</th>
<th>10:01AM, 06/16/2008</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bank processes</td>
<td>debit to 10035692876 in the amount of $17,000.00</td>
</tr>
<tr>
<td>Bank processes</td>
<td>debit to 10035692877 in the amount of $16,000.00</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Time now is</th>
<th>10:00AM, 06/17/2008</th>
</tr>
</thead>
<tbody>
<tr>
<td>Customer</td>
<td>bobma</td>
</tr>
<tr>
<td></td>
<td>resumes notifications on account 10035692877</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Time now is</th>
<th>10:01AM, 06/17/2008</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bank processes</td>
<td>debit to 10035692877 in the amount of $20,000.00</td>
</tr>
</tbody>
</table>

New notifications sent to customer bobma

<table>
<thead>
<tr>
<th>type</th>
<th>account</th>
<th>timestamp</th>
<th>amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>debit</td>
<td>10035692877</td>
<td>9:30AM, 03/18/2008</td>
<td>$15,000.00</td>
</tr>
<tr>
<td>debit</td>
<td>10035692876</td>
<td>9:30AM, 03/18/2008</td>
<td>$17,000.00</td>
</tr>
<tr>
<td>debit</td>
<td>10035692877</td>
<td>10:01AM, 06/17/2008</td>
<td>$20,000.00</td>
</tr>
</tbody>
</table>
Example: GUI for Manage Notifications Tx

- **User Interface implies specific functionality:**
  - List of accounts
  - Ability to make changes to notifications
  - List of active notifications
- **This functionality can be tested independently of UI**
**Single Transaction Test**

**Use Case: Manage Notifications**

<table>
<thead>
<tr>
<th>Customer</th>
<th>bobma</th>
<th>logs in</th>
</tr>
</thead>
</table>

System lists all available accounts for the authorized customer:

<table>
<thead>
<tr>
<th>account</th>
<th>type</th>
<th>notifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>10035692877</td>
<td>chequing</td>
<td>disabled</td>
</tr>
<tr>
<td>10035692890</td>
<td>savings</td>
<td>disabled</td>
</tr>
<tr>
<td>20010928892</td>
<td>credit line</td>
<td>disabled</td>
</tr>
</tbody>
</table>

Customer sets notification threshold for all transactions from all locations to $10,000.00 on account 10035692877 via email to bobma@live.com

**Side effect of Adding A Notification**

System lists all available accounts for the authorized customer:

<table>
<thead>
<tr>
<th>account</th>
<th>type</th>
<th>notifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>10035692877</td>
<td>chequing</td>
<td><strong>enabled</strong></td>
</tr>
<tr>
<td>10035692890</td>
<td>savings</td>
<td>disabled</td>
</tr>
<tr>
<td>20010928892</td>
<td>credit line</td>
<td>disabled</td>
</tr>
</tbody>
</table>

Notification settings for account 10035692877:

<table>
<thead>
<tr>
<th>transaction type</th>
<th>location where initiated</th>
<th>threshold amount</th>
<th>via</th>
<th>address</th>
</tr>
</thead>
<tbody>
<tr>
<td>all</td>
<td>all</td>
<td>$10,000.00</td>
<td>email</td>
<td><a href="mailto:bobma@live.com">bobma@live.com</a></td>
</tr>
</tbody>
</table>

**Medium Detail; Medium Scope**

Still no mention of User Interface!
Example:

**Business Rule Specs**

**Threshold per Charge Type**

**Configuration**

<table>
<thead>
<tr>
<th>Customer</th>
<th>Account</th>
<th>Label</th>
<th>Added()</th>
</tr>
</thead>
<tbody>
<tr>
<td>bobma</td>
<td>100372</td>
<td>Checking</td>
<td></td>
</tr>
</tbody>
</table>

**Process Transaction**

<table>
<thead>
<tr>
<th>Account</th>
<th>Amount</th>
<th>Charge Type</th>
<th>Notify?</th>
</tr>
</thead>
<tbody>
<tr>
<td>100372</td>
<td>Travel</td>
<td>999.99</td>
<td>No</td>
</tr>
<tr>
<td>100372</td>
<td>Travel</td>
<td>1,000.00</td>
<td>Yes</td>
</tr>
<tr>
<td>100372</td>
<td>Restaurant</td>
<td>99.99</td>
<td>No</td>
</tr>
<tr>
<td>100372</td>
<td>Restaurant</td>
<td>100.00</td>
<td>Yes</td>
</tr>
<tr>
<td>100372</td>
<td>Groceries</td>
<td>264.22</td>
<td>No</td>
</tr>
<tr>
<td>100372</td>
<td>Groceries</td>
<td>264.23.00</td>
<td>Yes</td>
</tr>
<tr>
<td>100372</td>
<td>Other</td>
<td>9.999.99</td>
<td>No</td>
</tr>
<tr>
<td>100372</td>
<td>Other</td>
<td>10,000.00</td>
<td>Yes</td>
</tr>
</tbody>
</table>

---

*High Detail; Narrow Scope
Completely ignores UI!*
Changing Level of Abstraction/Detail

- Need to Reduce Detail or Reduce Scope

<table>
<thead>
<tr>
<th>Workflow</th>
<th>Transactions</th>
<th>Business Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>Too vague</td>
<td>Too much detail</td>
<td>Unmaintainable</td>
</tr>
</tbody>
</table>

Reduce Detail a Lot
Reduce Both a bit
Reduce Scope a Lot!

Copyright 2011 Gerard Meszaros
Agenda

• Motivation
• Changing the Role of Test Automation
  • Approaches to Test Automation
    – Test Preparation Approach
    – Test Definition Language
    – Test Execution Interface
• Test Automation Strategy
Why is Automation Approach Important?

• **Common Failure Mode:**
  – Choose tools, then try to make them work
  – Wrong tools can prevent achieving goals

• **Better Approach:**
  – Choose automation approach to achieve goals
  – Then, select tools to support it
# Common Approaches to Test Automation

<table>
<thead>
<tr>
<th>Test Preparation</th>
<th>Test Language</th>
<th>Test Interface</th>
<th>Test Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recorded</td>
<td>Code</td>
<td>Raw UI</td>
<td>Global, Static</td>
</tr>
<tr>
<td>Refactored</td>
<td>Keyword</td>
<td>Adapter</td>
<td>Per Run</td>
</tr>
<tr>
<td>Handwritten</td>
<td>Data</td>
<td>API</td>
<td>Per Test</td>
</tr>
</tbody>
</table>

## Test Scenario Name

---

### Preconditions

1. Do Something
2. Check Something

### Clean Up

---

### How Prepared?

### How Expressed?

### How to Talk?

---

## Our System

- **Interface**
- **Business Logic**
- **Container**
- **Database**
- **Services**

## Fixture (state)

---

### How Set Up?

---

### How to Test?

---

---

---

---

---
### (C)OTS Record&Playback

<table>
<thead>
<tr>
<th>Test Preparation</th>
<th>Test Language</th>
<th>Test Interface</th>
<th>Test Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recorded</td>
<td>Code</td>
<td>Raw UI #</td>
<td>Global, Static</td>
</tr>
<tr>
<td>Refactored</td>
<td>Keyword*</td>
<td>Adapter</td>
<td>Per Run</td>
</tr>
<tr>
<td>Hand-written</td>
<td>Data</td>
<td>API</td>
<td>Per Test</td>
</tr>
</tbody>
</table>

**Notes:**
* Keywords, if used, tend to be very low level:
  * `GotoWindowNamed: name`
  * `SelectFieldNamed: name`
  * `EnterText: text`
  * (Not the same as true Keyword-Driven testing)

# Most COTS Tools operate at UI or HTTP interface; many open-source tools do so as well
Keyword-Driven Tests

- The tests are expressed in domain-specific vocabulary.
- The tests are read & executed by a test interpreter written by techies.

Prepared like Hand-Coded Tests but with a much more limited vocabulary.
# Keyword-Driven Tests

<table>
<thead>
<tr>
<th>Test Preparation</th>
<th>Test Language</th>
<th>Test Interface</th>
<th>Test Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recorded</td>
<td>Code</td>
<td>Raw UI *</td>
<td>Global, Static</td>
</tr>
<tr>
<td>Refactored</td>
<td>Keyword</td>
<td>Adapter</td>
<td>Per Run</td>
</tr>
<tr>
<td>Hand-written</td>
<td>Data</td>
<td>API</td>
<td>Per Test</td>
</tr>
</tbody>
</table>

**Notes:**
- While the Keyword Interpreter may go against the Raw UI, it is better to delegate to an adapter if no API is available.

```
<table>
<thead>
<tr>
<th></th>
<th>Poor</th>
<th>OK</th>
<th>Good</th>
</tr>
</thead>
<tbody>
<tr>
<td>Example Driven</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Legacy</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Workflow</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>System</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Business Rules</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Component</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unit</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>New</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Workflow</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>System</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Component</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Business Rules</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Unit</td>
<td></td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>
```
Sample Keyword-Driven Test
(e.g. Cucumber, JBehave, or Fit)

- Test script defined using keywords
- Keyword Interpreter invokes underlying code
- Can go direct to API or via an Adapter
Exercise 3 – Keyword-Driven Test

• Provide examples for the following workflow (Min. detail)

Note: Can assume an input queue exists for each role if that helps checking.
Data-Driven Tests

- The tests are expressed as tabular data by users.
- The tests are read & executed by a test interpreter written by techies.

Runs the same test script many times; once per set of data.
## Data-Driven Test

<table>
<thead>
<tr>
<th>Test Preparation</th>
<th>Test Language</th>
<th>Test Interface</th>
<th>Test Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recorded *</td>
<td>Code *</td>
<td>Raw UI</td>
<td>Global, Static #</td>
</tr>
<tr>
<td>Refactored</td>
<td>Keyword</td>
<td>Adapter</td>
<td>Per Run</td>
</tr>
<tr>
<td>Hand-written</td>
<td>Data</td>
<td>API</td>
<td>Per Test</td>
</tr>
</tbody>
</table>

**Notes:**

* The underlying script may be either hand-written or recorded and parameterized. But the data scenarios (input values and expected outputs) are almost always prepared by hand.

# The inputs/outputs are per test (per row) but there may be global or per-run data used as reference data by the underlying script.

<table>
<thead>
<tr>
<th></th>
<th>Example Driven</th>
<th>Legacy</th>
<th>New</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Poor</td>
<td>OK</td>
<td>Good</td>
</tr>
<tr>
<td>Workflow</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>System</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Business Rules</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Component</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unit</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Workflow</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>System</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Component</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Business Rules</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unit</td>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Much Ado About Agile 2011

Copyright 2011 Gerard Meszaros
Sample Data-Driven Test in FIT

PayrollFixtures.WeeklyCompensation

<table>
<thead>
<tr>
<th>Standard Hours</th>
<th>Holiday Hours</th>
<th>Hourly Wage</th>
<th>Pay( )</th>
</tr>
</thead>
<tbody>
<tr>
<td>40</td>
<td>0</td>
<td>10</td>
<td>$400</td>
</tr>
<tr>
<td>40</td>
<td>0</td>
<td>20</td>
<td>$800</td>
</tr>
<tr>
<td>41</td>
<td>0</td>
<td>20</td>
<td>$830</td>
</tr>
<tr>
<td>40</td>
<td>1</td>
<td>20</td>
<td>$840</td>
</tr>
<tr>
<td>41</td>
<td>1</td>
<td>20</td>
<td>$870</td>
</tr>
</tbody>
</table>

• Same script is run for each row of table
• Avoids duplication of test script.
• Compact summary of input values & results
• Sometimes called “Business Unit Test” or “Business Rule Test”
### Sample Data-Driven Test in FIT

**PayrolFixtures.WeeklyCompensation**

<table>
<thead>
<tr>
<th>Standard Hours</th>
<th>Holiday Hours</th>
<th>Hourly Wage</th>
<th>Pay( )</th>
</tr>
</thead>
<tbody>
<tr>
<td>40</td>
<td>0</td>
<td>10</td>
<td>$400</td>
</tr>
<tr>
<td>40</td>
<td>0</td>
<td>20</td>
<td>$800</td>
</tr>
<tr>
<td>41</td>
<td>0</td>
<td>20</td>
<td>$830</td>
</tr>
<tr>
<td>40</td>
<td>1</td>
<td>20</td>
<td>$840 \textit{expected} $800 \textit{actual}</td>
</tr>
<tr>
<td>41</td>
<td>1</td>
<td>20</td>
<td>$870 \textit{expected} $830 \textit{actual}</td>
</tr>
</tbody>
</table>

- Same script is run for each row of table
- Avoids duplication of test script.
- Compact summary of input values & results
- Sometimes called “Business Unit Test” or “Business Rule Test”
Exercise – Business Unit Test

• Rewrite the tests for the Invoice Total logic using a Data-Driven Business Unit Test that talks directly to the component that calculates the total.

• Focus on single-item invoices.
  – E.g. Each row describes the total expected for one line item.

• Suggested test cases are in the Testers’ Package

• You may use the template provided by “Test Automation” or you may invent your own.
Agenda

• Motivation
• Changing the Role of Test Automation
• Approaches to Test Automation
  – Test Preparation Approach
  – Test Definition Language
  – Test Execution Interface
• Test Automation Strategy
What Does It Take...?

• to be able to write tests like this?

• We need some technical skills to implement the “fixtures” or “interpreters” of our testing language, and either

• the right programming interfaces in the system, or

• we need to do extensive wrappering to simulate them
Keeping Tests Simple: Testing via API

What we want to write:

Test Invoice Generation
-New Customer

Logged in as Clerk
Item1, Item2 exist

1. CreateCustomer "Acme"
2. CreateAccount NewCust
3. AddPurchase Item1
4. AddPurchase Item2
5. GenerateInvoice NewAcct
6. ...

• API’s need to be designed in
  – Design for Testability

• Requires collaboration with Dev’t
  – Agile fosters collaboration through co-located teams
When There’s No API Available

Without a test API we have to write:

Test Invoice Generation
- New Customer

Goto Login screen
Enter "Clerk" in UserName field
Enter "Pw123Secret" in Password field
Enter ....

Goto Cust Screen
Click “New Customer”
Enter “Acme” in Name field
Enter “123 Main St.” in Addr field
Enter ....
GotoScreen( “Account” )
Find customer “Acme”
Click “Add Account”
Enter “Credit” in Type field
Enter ....

• Large gap between:
  – what we want to write & what can be executed
  – Many tests to adjust when UI changes \(\rightarrow\) High Maintenance Cost
Keeping Tests Simple: Testing via Adapters

What we want to write:

Test Invoice Generation
  - New Customer

Logged in as Clerk
Item1, Item2 exist

1. CreateCustomer  "Acme"
2. CreateAccount  NewCust
3. AddPurchase  Item1
4. AddPurchase  Item2
5. GenerateInvoice  NewAcct
6. ....

Adapters can be tacked on
  - Single place to adjust when UI changes
  - But may be complex and error prone

Copyright 2011 Gerard Meszaros

Much Ado About Agile 2011
Test - After Architecture

• Must test through User Interface

Workflow Test

System Under Test

- Configuration User Interface
- Configure Notification Threshold
- Process Transaction
- Should we Notify?
- Do Notification.

Notification Rules

Notification Log
Test-Driven Architecture

- Need to provide API’s to invoke functionality directly

Workflow Test

System Under Test

- Configure Notification Threshold
- Process Transaction
- Should we Notify?
- Do Notification.

Notification Log

Notification Rules

Configuration Interface

Transaction Interface
Test-Driven Architecture

Should we Notify?

Do Notification.

Configure Notification Threshold

Process Transaction

Notification Log

Configuration User Interface

Transaction Interface

Notification Rules

TX Test
Test-Driven Architecture

- With the right architecture, automating these tests is trivial
What About Legacy Systems?

• How can we get automated regression tests in place quickly?
Sample Recorded Test

@@ Login()
Browser("Inf").Page("Inf").WebButton("Login").Click
@@ GoToPage("MaintainTaxonomy")
Browser("Inf").Page("Inf_2").Check CheckPoint("Inf_2")
Browser("Inf").Page("Inf_2").Link("TAXONOMY LINKING").Click
Browser("Inf").Page("Inf_3").Check CheckPoint("Inf_3")
Browser("Inf").Page("Inf_3").Link("MAINTAIN TAXONOMY").Click
Browser("Inf").Page("Inf_4").Check CheckPoint("Inf_4")
@@ AddTerm("A","Top Level", "Top Level Definition")
Browser("Inf").Page("Inf_4").Link("Add").Click
wait 4
Browser("Inf_2").Page("Inf").Check CheckPoint("Inf_5")
Browser("Inf_2").Page("Inf").WebEdit("childCodeSuffix").Set "A"
Browser("Inf_2").Page("Inf").WebEdit("taxonomyDto.descript").Set "Top Level"
Browser("Inf_2").Page("Inf").WebEdit("taxonomyDto.definiti").Set "Top Level Definition"
Browser("Inf_2").Page("Inf").WebButton("Save").Click
wait 4
Browser("Inf").Page("Inf_5").Check CheckPoint("Inf_5_2")
@@ SelectTerm("[A]-Top Level")
Browser("Inf").Page("Inf_5").WebList("selectedTaxonomyCode").Select "[A]-Top Level"
@@ AddTerm("B","Second Top Level", "Second Top Level Definition")
Refactored Recorded Test

Login()

GoToPage("MaintainTaxonomy")

AddTerm("A","Top Level", "Top Level Definition")

SelectTerm("[A]-Top Level")
Refactored Recorded Test

Login()

GoToPage("MaintainTaxonomy")

AddTerm("A","Top Level", "Top Level Definition")

SelectTerm("[A]-Top Level")

AddChildToCurrentTerm( "A.1", "Definition of 1st Child Term of A")

AddChildToCurrentTerm( "A.2, "Definition of 2nd Child Term of A")

Now we hand-write additional tests using the resulting adapter (library)
Record, Refactor, Playback

- Use Test Recording as a way to capture tests
- Remove duplication by replacing with calls to domain-specific Test Utility Methods
  - using Extract Method refactorings
- Make Test Utility Methods reusable
  - Replace Hard-Coded Literal Values with variables/parameters
- Effectively turns recorded tests into programmed or keyword-driven test scripts
  - But, still through UI Adapter & original tool choice

Most appropriate with legacy systems
Especially with many interfaces
# Record, Refactor, Playback

<table>
<thead>
<tr>
<th>Test Preparation</th>
<th>Test Language</th>
<th>Test Interface</th>
<th>Test Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recorded</td>
<td>Code</td>
<td>Raw UI</td>
<td>Global, Static</td>
</tr>
<tr>
<td>Refactored</td>
<td>Keyword</td>
<td>Adapter #</td>
<td>Per Run</td>
</tr>
<tr>
<td>Hand-written</td>
<td>Data</td>
<td>API</td>
<td>Per Test</td>
</tr>
</tbody>
</table>

**Notes:**

# The result of refactoring is an adapter between the test script and the SUT's UI.
Built-In Record & Playback

- User executes tests manually; SUT records as tests
- Tool replays tests later without user intervention

The tests are data interpreted by the test runner.
Built-in Record&Playback

<table>
<thead>
<tr>
<th>Test Preparation</th>
<th>Test Language</th>
<th>Test Interface</th>
<th>Test Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recorded</td>
<td>Code</td>
<td>Raw UI</td>
<td>Global, Static</td>
</tr>
<tr>
<td>Refactored</td>
<td>Keyword</td>
<td>Adapter</td>
<td>Per Run</td>
</tr>
<tr>
<td>Hand-written</td>
<td>Data</td>
<td>API</td>
<td>Per Test</td>
</tr>
</tbody>
</table>

Notes:
- Needs to be implemented within SUT
- Can sometimes be retrofitted to legacy systems

Most appropriate with legacy systems when playing “automation catch-up”
Sample Built-in R&PB Test Recording

## 2. Supply Create

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Type</th>
<th>Used Value</th>
<th>Default or Choices Value(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>select-supply</td>
<td>selection</td>
<td>Create train</td>
<td>Create train</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Create gang</td>
</tr>
<tr>
<td>rtc-initials</td>
<td>output</td>
<td></td>
<td>HDM</td>
</tr>
<tr>
<td>engineno</td>
<td>input</td>
<td>9595</td>
<td>DIRECTIONAL</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>WORK</td>
</tr>
<tr>
<td>designation</td>
<td>selection</td>
<td>DIRECTIONAL</td>
<td>ENG</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>PSGR</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>MIXED</td>
</tr>
<tr>
<td>direction</td>
<td>selection</td>
<td>NORTH</td>
<td>SOUTH</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>NORTH</td>
</tr>
<tr>
<td>shortcode</td>
<td>output</td>
<td>X 9595 N</td>
<td>ignore</td>
</tr>
</tbody>
</table>
### Raw XML for “Designation” Field

```
<field name="designation" type="selection">
  <used-value>DIRECTIONAL</used-value>
  <expected>
    <value>DIRECTIONAL</value>
    <value>WORK</value>
    <value>PSGR</value>
    <value>PLOW</value>
    <value>PLOW WORK</value>
    <value>ENG</value>
  </expected>
  <actual>
    <value status="ok">DIRECTIONAL</value>
    <value status="ok">WORK</value>
    <value status="ok">ENG</value>
    <value status="ok">PSGR</value>
    <value status="surplus">MIXED</value>
    <value status="ok">PLOW</value>
    <value status="ok">PLOW WORK</value>
  </actual>
</field>
```
Sample R&PB Test Hooks

```c
choice = display_dialog(choices_list, row, col, title, key);
```
Sample R&PB Test Hooks

```c
choice = display_dialog(Choices_list, row, col, title, key);

if (recording_is_on())  {
    record_choice(dialog_id, choice_list, choice, key);
}
```
Sample R&PB Test Hooks

if (playback_is_on()) {
    choice = get_choice_for_playback(dialog_id, choices_list);
} else {
    choice = display_dialog(choices_list, row, col, title, key);
}

if (recording_is_on()) {
    record_choice(dialog_id, choice_list, choice, key);
}
## Hand-Coded Tests

<table>
<thead>
<tr>
<th>Test Preparation</th>
<th>Test Language</th>
<th>Test Interface</th>
<th>Test Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recorded</td>
<td>Code</td>
<td>Raw UI</td>
<td>Global, Static</td>
</tr>
<tr>
<td>Refactored</td>
<td>Keyword</td>
<td>Adapter</td>
<td>Per Run</td>
</tr>
<tr>
<td>Hand-written</td>
<td>Data</td>
<td>API</td>
<td>Per Test</td>
</tr>
</tbody>
</table>

### Notes:
- Hand-written code requires software development skills and test automation skills.
- API preferred but can script browser-based (UI) tests.
- Code can be primitive or abstract therefore...
  - Developers need training on writing clear tests!

### Example Driven

<table>
<thead>
<tr>
<th>Legacy</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Workflow</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>System</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Business Rules</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Component</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unit</td>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>New</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Workflow</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>System</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Component</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Business Rules</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unit</td>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Changing the Role of Testing

**Requirements**

- Business Facing
  - Acceptance Tests
  - Regression Tests
- Technology Facing
  - Unit Tests
  - Component Tests

**Define Product**

- Unit Tests
- Component Tests

**Critique Product**

- Usability Tests
- Exploratory Tests
- Property Tests
  - (Response Time, Security, Scalability)

**Report Card**

- Functionality: B
- Usability: C
- Scalability: A
- Response: B
- Availability: C

For effective prevention:

1. Tests must be available before development
2. Developers must be able to run tests before check-in

Thanks to Brian Marrick and Mary Poppendieck
Preventing Coding Defects
(Building the Product Right)
Preventing Coding Defects
(Building the Product Right)

Write Req'ts
Review Req'ts
Update Req'ts
Sign Off on Req'ts
Design Software

Write Unit Tests
Write Code
Run Unit Tests
Identify / Debug
Determine Fix

Deploy to QA
Test Application
Deploy System

Copyright 2011 Gerard Meszaros
Preventing Coding Defects
(Building the Product Right)

- Write Req’ts
- Review Req’ts
- Update Req’ts
- Sign Off on Req’ts
- Design Software

- Write Unit Tests
- Write Code
- Run Unit Tests

(Unit) Test-Driven Development

Prevent defects in new code

Prevents bugs crawling back in
public void testAddItemQuantity_severalQuantity () {
    // Setup Fixture
    final int QUANTITY = 5;
    Address billingAddress = new Address("1222 1st St SW", "Calgary",
            "Alberta", "T2N 2V2", "Canada");
    Address shippingAddress = new Address("1333 1st St SW", "Calgary",
            "Alberta", "T2N 2V2", "Canada");
    Customer customer = new Customer(99, "John", "Doe", new BigDecimal("30"),
            billingAddress, shippingAddress);
    Invoice invoice = new Invoice(customer);
    // Exercise SUT
    invoice.addItemQuantity(product, QUANTITY);
    // Verify Outcome
    List lineItems = invoice.getLineItems();
    if (lineItems.size() == 1) {
        LineItem actualLineItem = (LineItem) lineItems.get(0);
        assertEquals(invoice, actualLineItem.getInvoice());
        assertEquals(product, actualLineItem.getProduct());
        assertEquals(quantity, actualLineItem.getQuantity());
        assertEquals(new BigDecimal("30"),
                actualLineItem.getPercentDiscount());
        assertEquals(new BigDecimal("19.99"),
                actualLineItem.getUnitPrice());
        assertEquals(new BigDecimal("69.96"),
                actualLineItem.getExtendedPrice());
    } else {
        assertTrue("Invoice should have exactly one line item", false);
    }
}
Hand-Coded Test – Appropriately Abstracted

```java
public void testAddItemQuantity_severalQuantity () {

    // Fixture set up:
    final int QUANTITY = 5 ;
    Product product = createAnonymousProduct();
    Invoice invoice = createAnonymousInvoice();

    // Exercise SUT
    invoice.addItemQuantity(product, QUANTITY);

    // Verify
    LineItem expectedLineItem = newLineItem( invoice,
                                              product, QUANTITY, product.getPrice() * QUANTITY );

    assertExactlyOneLineItem( invoice, expectedLineItem );
}
```

Developers need training on effective unit testing!
Agenda

• Motivation
• Changing the Role of Test Automation
• Approaches to Test Automation

• Test Automation Strategy
  – Selecting the right Approach(es)
  – Maximizing Automation ROI
So What’s the Point?

Why is the approach to test automation significant?

Because test automation is hard work

And the approach effects the nature of the benefits of the automation.
How Effective is our Automation?

• Are the tests fully automated?
  – Can they run unattended?
  – Are they fully self-checking?

• Are the tests low maintenance?
  – How often do we need to adjust them?
  – How many tests are affected by a change in the SUT?

• Do the tests describe the requirements clearly?
  – Can everyone understand them?
  – Could we (re)build the system from them?

• Can anyone run them?
  – Can developers run them before checking in code?
For Success, Focus on Intent

• Choose the approach first, then pick tools
  – Tools must support the approach chosen

• Write the tests using the best language for expressing the requirement being validated.
  – Not necessarily the language provided by the System Under Test’s interface
  – May require different approaches for different tests

• Close any gap using an adapter if necessary
Which Automation Approach?

Depends heavily on Context

• **Legacy Systems:**
  – Stabilize with Recorded Tests while you refactor to enable Component testing.
  – Only do hand-written unit tests for new components.

• **Greenfield Development:**
  – Keyword-Driven workflow and system tests.
  – Data-Driven tests for business rules
  – TDD via hand-written Unit Tests
Which Automation Approach?

• **Recorded tests:**
  – implies a “Test After” approach; won’t help define the requirements
  – Typically results in tests with Primitive Obsession
    ➔ Fragile Tests with high test maintenance cost
  – Best for: Playing “Catch-up” on Legacy Systems

• **Hand-Written Tests:**
  – Amenable for use in Example-Driven Development
    » But must use Domain-Specific terminology to be effective
  – Can be written in code or keywords depending on who’s preparing the tests
  – Best for: Workflow tests (Keyword) and unit tests (code)
Which Automation Approach?

• **Keyword-Driven Tests:**
  – Good separation between business and technical work involved in automating tests.
  – Easy to prepare before development.
  – Best for expressing workflow or system tests.

• **Data-Driven Tests:**
  – Best for repeating same test script with many combinations of inputs
  – Best for: Verifying Business Rules & Algorithms
    » (A form of Component Testing)
Maximizing Test Automation ROI

• Need to Treat Automation as an Investment
• Need to Prioritize / Triage Which Tests to Automate
• At least 3 Approaches to Choose From:
  – Traditional QA-Based “Test After” Automation
  – Collaborative Critical Path Automation
  – Collaborative Selective Automation
Automation After Dev Complete
A.K.A. Traditional Approach to Automation

Summary:
– Done by QA/SV Department (i.e. Testers)
– After Product is Built
– Typically done using (C)OTS Record & Playback tools

Issues:
• Too Late for Defect Prevention
  – Tests aren’t available to development team
• Too Late to Ensure Easy Automation
  – System not Designed for Testability
• Tools Create Fragile Tests
  – Unreadable due to Primitive Obsession and too much duplication
Collaborative Automation on Critical Path
A.K.A. Dogmatic (A)TDD Approach

Summary:
– Goal: 100 % automation
– Automate Tests Before Building Functionality
  » Test automation task for each User Story

Issues:
• Some Tests are MUCH Harder to Automate
• May Increase Costs and Delay Benefits of Functionality
• May Cause EDD to be Abandoned
Collaborative Automation based on ROI
A.K.A. Pragmatic Approach

Summary:
– Goal: Just Enough Automation
– Apply Agile Principles to Implementation of Automation

Issues:
– Won’t Have Complete Test Coverage
– Can Lead to Automation Being Dropped in Favour of More Functionality
  – Requires a Disciplined Product Owner, or,
  – A Fixed Budget for the Automation
What if Automation is Really Hard?

- Apply the 80/20 rule
- Define the tests first
- Automation is optional

**User Goal**

**Feature**

**Story Title**

**Story Narrative**

**Story Scenarios**

**Define Acceptance Criteria**

**Make Concrete Add Data**

**Story Example**

**Executable Example**

**Product Development**

**Satisfied Example**

Significant Value in Providing Examples / Tests Before Development
Closing Thoughts

• Are you automating to find defects or prevent them?

• Are your automated tests good examples?
  – Why not? What would you need to change?

• Are your tests low maintenance?
  – Why not? What causes them to break?
  – What could you change to make them break less often?
  – .... to reduce the impact of breakage?

• Can anyone run the tests at any time?
  – Can the developers run the tests on-demand before they check their code in?
  – What would you have to change to make that possible?
Thank You!

Gerard Meszaros
Agile2011ATAS@gerardm.com
http://www.xunitpatterns.com

Slides:
http://Agile2011ATAS.xunitpatterns.com

Call me when you:
• Want to transition to Agile or Lean
• Want to do Agile or Lean better
• Want to teach developers how to test
• Need help with test automation strategy
• Want to improve your test automation
References

• For Success, Build Record/Playback into Your Application - StarEast 2008 Class
  – http://builtinrecordandplayback.xunitpatterns.com

• These Slides:
  – http://strategy.testAutomationPatterns.com