Testing and Quality in Agile Development

Speaker: Allan Watty
Company: ABB Inc
Website: www.abb.com/enterprise-software

Welcome to the PMI Houston Conference & Expo 2015

• Please put your phone on silent mode
• Q&A will be taken at the close of this presentation
• There will be time at the end of this presentation for you to take a few moments to complete the session survey. We value your feedback which allows us to improve this annual event.
Key Objectives

Will cover:

- Challenges to expect with Agile test practices in large-scale, multi-team projects
- The Power of three
- Team dynamics
- Organizational strategies & considerations
- Test automation considerations
- Customer, business impacts
Key Objectives

Will not cover

- Automation details
- In-depth tool details
- Code samples
Speaker Bio

ABB Inc., 2012 – present
• Senior Development Manager
• New Product development with 2 Scrum Teams
PROS, 2005 – 2012: Development Manager, QA Manager

Education
• B.S. Math & Biology, Lehman College, CUNY
• M.S. Computer Science, NYU Polytechnic
• M.B.A, University Of Houston Bauer College,
Certifications: PMP, CSM

President – Agile Leadership Network Houston Chapter
Former Life: Musician, Math Teacher
Agenda

- Agile Testing Challenges
- The whole team approach
- Designing for Testability
- Definitions of Ready and Done
- Risk based testing
- Using the Power of Three
- Making testing a first-class citizen
- Acceptance Test Automation
- Core Testing Practices
- Use Metrics to reflect and adapt
Key Challenges on Agile teams

- Mini-Waterfalls
- Testers are always behind
- Short-cuts
- Cone of Uncertainty
- Clarity on requirements
- Huge Manual Regression Backlog
- Poor estimates
- What should be automated?
- Late changes to the design
- Applications not designed for testability
- Testing at the end does not work
- Moving / unclear definition of “DONE”
Trouble Ahead?

- Lengthy Code, Test and Fix cycles
- High % of unfinished stories
- Debate on requirements after story is implemented
- Stories accepted with defects
- Broken builds is treated as no big deal
- Numerous build warnings
- Code Coverage is lowered by new code
- Stories started without clarity on success criteria
- Inability to Automate Acceptance Tests within Sprint
Cone of Uncertainty*

Variability in the Estimate of Project Scope effort, cost, features

Time

Product Definition
Detailed Requirements
User Interface Design

*Steve McConnell, Software Estimation: Demystifying the Black Art
The Whole Team Approach

• Everyone is committed to deliver the highest quality software possible
• Daily collaboration
• Testing is a central practice
• Embedded testers and developers
• Task and knowledge sharing

• Release & Sprint planning
• Start and finish together
• Common definition of “READY”
• Common definition of “DONE”
Foundation of A Great Team

Patrick Lencioni: Overcoming the Five Dysfunctions of a Team
Design for Testability

• Ensure that any new features/code that is implemented can be tested.
• SOLID Design principles
• Layered Architecture
• Interface Programming
• User Behavior
• API Contract
• “How can you know when I am done?”
Using the Power of Three

• Developers, QE, Product owner share ownership of the story.
• Shepherd a Story/Feature from inception to completion
• Transparency and team work
• Shared understanding of:
  ❑ What is being built
  ❑ Conditions of success
• Nothing is thrown over the wall
• Work at a sustainable pace
Make testing a first-class citizen

- Test scripts get the same care and feeding as production code
- Failed tests results get actioned.
- Use Smoke Tests during development to detect DOA builds
- Test Framework must be maintainable by DEV and QE
- Test Code is stored in CM repository
- Broken automated test will slow new feature work.
Definitions of Ready and Done

• Story Readiness
  o What are the goals of pre-grooming and grooming?
  o When is a story ready to be brought into a sprint?

• Story Completeness
  o When is a story done?
  o What are all the things that should be completed before we accept a story?

• Definitions must be common across teams.
  o A story is not done until all its acceptance tests pass
Definition of Done

Key Practice: Teams should agree on “DONE” at the start of the release.
• Without a consistent meaning of done, velocity cannot be estimated.
• Ensures that we deliver a potentially shippable product
• You are not finished with a feature until it meets the benchmark defined by done.

•Simple verifiable definition:
  o Coding is Complete
  o Testing is Complete
  o Related User Documentation is Complete
  o Feature is accepted by Product Owner
Why High Performance Agile Testing?

- Quality as a competitive advantage
- Overcome challenges of Agile testing
- Build capability to handle disruptive technology
- Be ready to scale as company grows
Look at the Big Picture

Use the Whole Team Approach

Risk Based Testing

Adopt an Agile Testing Mindset

Core Testing Practices

Automate Regression Testing

Collaborate with the Customer

Handle Technical Debt

Regular constant feedback

Key Success Factors
What is Agile Testing?

Before Agile:
Testing begins at the end

Predictive schedule

Adaptive schedule

Agile: Each story is developed and tested within an iteration
What is Agile Testing?

- Testing starts early and is continuous
  - Each story slice is tested as soon as its finished
  - Application is exercised constantly, no surprises later

- QE is part of, and embedded in the product team
  - Co-located product teams
  - Power of 3: developers, testers, product owners

- Team Focus:
  - Test Automation
  - Exploratory testing
  - Design for testability
  - Business Value
QA/QE Involvement

- Release Planning
  - Story mapping, Feature/Epic review
- Sprint Planning and Reviews
- User Story conditions of success
- Test Case maintenance
- Customer usage
- Risk assessment
- Metrics
- Design reviews
- Root Cause Analysis
- Test Frameworks
Look at the Big Picture

- Use business-facing tests, examples to drive development
- Use real world test data
- Be aware of impacts on other areas:
  - Software dimensions
  - Feature usage
  - Estimation
  - Focus on business value
  - User Stories
  - Understand your context
- Value of Testing: Shipping a product customers love
- Value of Bug Reports: Describing the actual problem
Each of these five dimensions can take one of three roles on any given project: a *driver*, a *constraint*, or a *degree of freedom*. 

```
Features

Staff

Schedule

Cost

Quality
```
Actual use of software features

Source: The Standish Group CHAOS Research
Agile Testing Mindset

• Testing is not a phase!
• Early feedback before feature is complete
• Key testing tasks: automation, exploratory
• One process – no handoffs
• Focus on conditions of satisfaction
• No Quality Police
• Pro-active approach
• Eliminate Waste
• Adding value
Adopt Regular Release Cadence
Handle Technical Debt

- Using a design strategy that isn't sustainable in the longer term, but yields a short term benefit.
- Taking short cuts on coding and testing
- Design flaws
- Messy code
- Poor functional test coverage
- Quality debt
Handle Technical Debt

Reckless

“We don’t have time for design”

Prudent

“We must ship now and deal with consequences”

Deliberate

“Iadventent

“What’s layering?”

“Now we know how we should have done it”
Handle Technical Debt

Key Practice:
• We will ship on time and revise the quick fix as our top priority in the next release.
• Must have commitment to schedule prioritize, and fix soon

“Take my advice: go well, not fast. Care about your code. Take the time to do things right. In software, slow and steady wins the race; and speed kills.”
Robert C. Martin (Uncle Bob)
Risk based testing

- Utilize support incidents and Root Cause Analysis output as additional test ideas.
- Review the likely use of each feature by customers.
- Analyze the likelihood and the impact of failure of key features.
- Prioritize regression testing based on risk
- Surround key functionality with unit and systems tests
Why Automate?

- Free up time for other important work
- Capture Repeatable tasks
- Safety net
- Quick feedback
- More time for exploratory testing
- Reduction of regression test time
- Tests provide living documentation
- You know what your system does
Regression Test Automation issues

- Initial hump of pain
- Fear of tool or code
- Test automation is software development!
- UI based testing vs. constant UI behavior changes
- Behind the UI tests requires development collaboration
Test Automation pyramid

- Manual Tests
  - QTP, eggPlant, Sikuli, Selenium, Watir

- GUI Tests
  - FitNesse, Cucumber, Specflow, Concordion, shell scripts

- Acceptance Tests (API Layer)
  - JuNit, Nunit, Flexunit, Klocwork, PMD, FindBugs

- Unit Tests / Component Tests
Story Workflow

1. Story Start
2. Story Review (Dev, QE, PM)
3. Acceptance Criteria Agreement?
   - Yes
     - Story Implementation
       - Create Feature Files
       - Write Failing Specflow Tests
       - Lightweight Design
       - Reviews
       - Incremental Testing
       - Exploratory Testing
       - Implement Story using TDD
       - Refactor
     - Acceptance Test Passing
       - Yes
         - Exploratory and manual tests
         - Story Review and Acceptance
         - Story Complete
       - No
         - Exploratory and manual tests
         - Story Review and Acceptance
         - Story Complete

The whole team handles implementation.
Testing the Windows Calculator

Types of testing
1. User Behavior
2. Calculator engine

How many times do you need to test the UI?
Behind the UI testing

Test all the services of an application separately from its user interface

Test cases

Fixture

System

Test cases

Fixture

Customer s, PM, Dev, QE

Development

PMI Houston Conference Expo 2015
Automated Acceptance Tests

• Automated Acceptance Tests
  o ATDD, BDD, Specification by example
• ATDD is about **communication**
• ATDD is a practice for collaboratively discussing acceptance criteria with examples.
• Like TDD, involves creating tests before code
• Tests represent expectations of behavior the software should have.
• Acceptance tests provide feedback to the team about how close we are to the “Definition of Done”
• Provides living documentation of your application
As a [Shopper] I want [to put items in my shopping cart] so that [I can manage items before I check out]

Acceptance Criteria N

Given I'm a logged-in User

When I go to the Item page
    And I click "Add item to cart"

Then the quantity of items in my cart should go up
    And my subtotal should increment
    And the warehouse inventory should decrement
Unit tests using TDD

Test Driven Development (TDD) steps:
1. Write a unit test for a new piece of code
2. Execute the unit test
3. It should fail ✗
4. Write the code that implements the function
5. Execute the unit test, it should now pass ✓
6. Refactor
7. Repeat at step 1
ATDD cycle

ATDD Cycle
ATDD cycle

1. Write Automated Acceptance Tests (AATs) using the acceptance criteria.
2. Test will fail initially. To make the AATs pass, start providing the functionality.
3. Now we need apply TDD concepts, write a failing unit test.
4. We write enough code to pass failed unit test.
5. Repeat at step 2, continue following UTDD
6. When a story is complete, AAT's and Unit Tests are passing, we now continue to step 1 again for the next story.
Behind the UI and API level automation

• Many tools are available to help with automation:
  o Specflow, Jbehave, Fitnesse, Concordion, Cucumber

• Tools are used to bridge the communication gap between domain experts, testers and developers.

• Bind business readable behavior specifications and examples to the underlying implementation.

• Specifications/user stories can be written in Gherkin format

• Tools enable separation of:
  o Input data and test scenarios (Feature Files)
  o Test Methods or Fixtures that that drives the application
  o Reporting of the results in HTML format
1. SpecFlow binds business readable behavior specifications and examples to the underlying implementation.

2. Lets us write specifications/user stories in plain readable Gherkin format.

3. Feature file: Specflow uses feature file to store the acceptance criteria of the features (User Stories)

4. Every scenario consists of a list of steps, which must start with one of the keywords Given, When, Then, But or And.

5. A test method is generated for each scenario step, the method name will be derived from the title of the scenario.
Selenium: UI level automation

• A set of tools that supports rapid development of test automation for web-based applications
• Support for a number of programming languages: Java, C#, Perl, PHP, Python, Ruby.
• Cross browsers support: IE, Firefox, Opera, Safari and Google Chrome
• Cross platform support: Windows, Linux and Macintosh
Test Automation

1. Multi-Layered Approach
2. Test Driven Development delivers unit, component tests
3. Acceptance Test Driven Development delivers API level, GUI tests
4. GUI smoke tests as regression safety net
Adopt Test Design Principles

- Single Purpose
- DRY
- Domain Specific Language
- Abstract Code out of Tests
- Setup and Teardown Tests
- Independence
- Tests must run green – all the time
- Common Test Standards (including naming conventions)
- Test code is production code
UI Tests

- Minimal and purposeful
- Workflow tests
- UI Smoke Test
- Page Object Pattern
- Push for API/Service level tests
Page Object Pattern

- Separates test specification from the test implementation
- Wraps an HTML page, or fragment, with an application-specific API
- Enables easy manipulation of page elements
- UI Smoke Test
- Encapsulate the mechanics required to find and manipulate the data in the gui control
- Utilize Data Driven Testing
Page Object Pattern
Development Practices

Key practices we have embraced:
- Before any code is written, acceptance criteria for a story is reviewed together by Product Owner, Developer and QE.
- For new functionality, the developer creates a light weight design and a design review is done.
- Development starts with writing unit tests.
- SpecFlow Acceptance Tests scenarios are written by QE.
- SpecFlow Acceptance Tests code is written by developers and based on scenarios from QE.
- Developers strive to give QEs testable functionality early and often.
- Code review *before* source code check-in.
- Apply any necessary changes due to code reviews.
- Continuous Integration builds are started after every check in.
- QE modifies Selenium UI tests if needed.
- Changes are tested and approved by QEs and Product Owners.
- Story is accepted by Product Owners.
Continuous Integration Practices

- Key practices we have embraced:
  - Continuous Integration builds are started after every check in
  - All unit tests are run at the beginning of a build
  - All automated acceptance tests are run at during the build
  - Any unit or acceptance test failures stop the build
  - Selenium UI smoke tests are executed daily
  - Selenium UI regression tests are executed twice a week
  - Any Selenium test failures are reported as defects and fixed within the sprint
  - Installation packages are built as part of CI
  - New installation is deployed to QA Development site as part of CI
  - New installation is manually deployed to QA Test site
Build Quality In

- Find and fix defects the moment they occur.
- Mistake-Proof the Process
- Think of tests as specifications. Use them to establish confidence in the correctness of the system at any time.
- Integrate Early and Often
- Don't Tolerate Defects
- If you expect to find defects during final verification, your development process is defective.
Stop The Line

- Broken Builds
- Warnings threshold exceeded
- Defect threshold exceeded
Core Testing Practices

- Review Stories and Acceptance tests with PM and Developers
- Test estimates are part of sprint backlog
- Design test cases:
  - Acceptance tests
  - System tests: Negative, boundary, i18n, equivalence tests
- Test case reviews with PM and Developers
- Work with developers to get testable chunks delivered early
- Test case automation
Core Testing Practices

• Regular Root Cause Analysis Sessions
• Developers execute acceptance tests before final story testing.
• Exploratory testing
• Performance and Scalability Testing
• Test Case Maintenance
  o Remove dead tests
  o Update and add new tests
• Automate acceptance tests during the sprint
• UI automated smoke test
• UI Workflow automated tests
Use Metrics to reflect and adapt

1. Percentage of Story points claimed
2. Percentage of test cases automated
3. Percentage of code coverage
4. Current # of Build Warnings
5. Number of defects found in the sprint
6. Number of defects found during regression
7. Number of defects fixed
8. Number of Open defects

Display metrics in each sprint review
Team Metrics

COG-PSM Sprint Metrics

- Points Planned
- Points Achieved
- Open Defects
- Code Coverage
Collaborate with Customers

“Customer involvement is the most critical factor in software quality “ *

- Learn how customers use the product
- Formal beta releases (Early Access Program)
- Formal means: tested features, feedback expected, partnership
- Include key customer workflows in testing
- Dev/QE/PO customer communication and visits
- Ask customers for concrete examples, scenarios

* Karl Wiegers, author - Software Requirements
Regular constant Feedback

- Use feedback to improve
- Make process / strategy corrections as needed
- Immediate Feedback: Continuous Integration & Deployment
- Learn from the tests
- Retrospectives
- Regular Root cause analysis of customer incidents
- Use support incidents to drive defect prevention
- Create taxonomy of defects
- Automate test cases from support incidents
Agile Testing: Key Success Factors

• Use the whole-team approach
• Adopt an Agile testing mind-set
• Automate regression testing
• Provide and obtain feedback
• Core Testing Practices
• Manage Technical Debt
• Collaborate with customers
• Look at the big picture
• Regularly reflect and adapt
Products that delight our customers!

• Under Commit!
  o Plan less work than you think you can do
  o Focus on finishing one user story at a time
  o 100% full means we do not go anywhere

• Budget time to
  o Learn new skills, make improvements
  o Refactor automated tests

• Focus on Mission:
  o Finding defects as early as possible in every release.
  o Quality becomes a competitive advantage
  o One team culture of developers, testers and product managers
Key References

- Agile Testing, by Lisa Crispin
- Essential Scrum, by Kenneth Rubin
- ATDD by Example: A Practical Guide to Acceptance Test-Driven Development, by Markus Gärtner
- Clean Code, by Robert Martin
- Lean-Agile Acceptance Test-Driven Development, by Ken Pugh
- Specification by Example by Gojko Adzic
- BDD in Action by John Ferguson Smart
- The Five Dysfunctions of a team by Patrick Lencioni
- Managing Software Debt, by Chris Sterling
Agile Leadership Network – Houston Chapter

- Venue: Sysco Corporation, 1390 Enclave Parkway.
- Meetings are held the 3rd Thursday of each month unless otherwise noted on the chapter’s website.
- Meeting format:
  - 6:00pm – 6:30pm – Network with your fellow Agile practitioners
  - 6:30pm – 8:00pm – Announcements and introductions, followed by our Featured Program
- Admission is Free
- Registration: http://alnhouston.org
Questions