Agile testing for a Waterfall World

Planning, preparing and executing tests can be challenging enough on a new project. There are expectations to manage, risks to assess, environments to understand and domains to investigate, while juggling client concerns and perceptions. Working with evolving web technologies adds to the fun. Now throw in a decision to do the project as a proof of concept Agile effort when it is part of a large program of conversions in a waterfall based shop, and you have a lot of angles for any tester to cope with.

How do you bring in enough traditional testing approaches and techniques to satisfy the client while engaging the project team as a fully contributing Agile-savvy technical tester? How do you need to adjust your use of techniques and approaches to work most effectively with an Agile team?

This presentation will examine the design and evolution of testing on a small java-based web application. The project technical environment includes the use of Ajax, Spring, and Hibernate for run-time implementation. The developers are doing extensive unit testing, using several tools including JUnit, HTTPUnit, and DBUnit, and are relying on QA to build out the more complex integration tests. However, since the requirements are intentionally only being fleshed out as the project progresses, fully detailed scripts for tests cannot be fully defined upfront. The technical implementation is changing over time as the client moves forward with other parallel conversion projects, so an extensive automated test suite may be rendered useless in our short two week iterations unless it can be readily adapted to the changes.

These challenges have led to the need to examine the underlying assumptions of each area of testing to determine which are impacted by this internally Agile / externally Waterfall project. For example, due to the timing of adding QA staff to the project, the first incremental release of the application went from unit testing to user acceptance testing (UAT), with no chance for more than a few hours of cursory QA review. Observers at the UAT session included the project architect and the UI designers along with key business stakeholders. We created an environment and process that was just structured enough for business comfort, yet open enough to enable the dialog and interaction sought by the developers. The result was a highly successful session that built client confidence in the team, our development approach, and the application. To achieve our objectives, we teased apart components of traditional UAT approaches to see where Agile preferences for direct communication between developers and users could be layered in while retaining a structure that would satisfy waterfall sensibilities.

This pattern of evaluating the structure of approaches and techniques to interweave diverse goals and needs will be ongoing throughout the project. It is anticipated that some pieces will work well while others will be, at best, learning experiences. The focus of the presentation will be to review what was attempted, actually done, and what we got for results, with evaluation of possible reasons for both our successes and failures.

Original (January) submission note: At this time, the project is only in the 2nd increment and I have been on the project for 3 weeks. The full range of manual and automated testing activities will be in place by mid- February (increment 3), with several increments being completed by
the end of April (increment 5). We are scheduled for implementation in July, so the majority of
the project will have been completed by the time of the presentation.

April update: We are now through three full rounds of UAT tests, 5 rounds of session based
tests, and 2 rounds of automated integration smoke tests. The UAT process we defined is being
adopted for use with other application modules within this development program based on our
success with it. The application functionality is approximately 40% complete, with an
estimated implementation date of mid-October. This is a slip of 10 weeks from our original
estimate due to delays in technical policy decisions external to the project on the client side and
challenges in getting sufficient time with subject matter experts for good user story definition.
However, despite the slip in the project completion date, we have learned quite a bit about what
is working and could be improved in our testing.

In addition to our experience with user acceptance, session based, and automated integration
tests, by July we also expect to have implemented some forms of database conversion and
stress testing in order to address emerging risks around a separate database conversion project
on which we are dependent.
Introduction

- Project Background: Agile development for a Waterfall client
- Review of primary assumptions made for Waterfall and Agile approaches
- Implications for testing of using an Agile approach
- Resulting test design process used
- Review of techniques chosen
- Discussion of adaptations made
- Results and Lessons Learned
Project parameters

- Large, government funded system of higher educational institutions
- Back office support system currently on “green screen” style system
- Part of a large program of application conversions
- Consultants make up all 7 developers, 2 BAs, 1 QA and 1 PM to the client’s 2 (part time) BAs, 1 PM and numerous technical and business stakeholders
- Geographically dispersed user community, with limited direct access

Client background

- Traditional waterfall style development process
- Relatively new Program Management Office (PMO)
- Understaffed IT department
- Record of slipped schedules and user-adapts-to-system style applications
- Little formal testing, typically done by BAs or small number of super-users
- Testing is generally confirmatory (“happy path”)
- History of large volume of bug reports when systems go live
Common Waterfall Assumptions

- Big Bang requirements
- Big Bang testing
- Structured, fully documented scripts
- May use well-documented exploratory techniques for some portions
- May be done by business analysts or users, focusing on “happy path” testing

- Separate business analyst, quality, and development teams
- User is at arms-length
- QA is viewed as gate-keeper for project
- User acceptance testing can be ineffectual, catastrophic, or both due to time pressures and high cost of change
—we are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more.

Some principles behind the Agile Manifesto

- Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
- Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
- Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
- Business people and developers must work together daily throughout the project.
Some principles behind the Agile Manifesto

- The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
- Continuous attention to technical excellence and good design enhances agility.
- Simplicity—the art of maximizing the amount of work not done—is essential.
- The best architectures, requirements, and designs emerge from self-organizing teams.

Early and Continuous Delivery of Working Software

- Requires testing to be ready EARLY
- Requires testing to be constantly in synch with changing code and requirements
- Requires frequent cycles of testing
- Requires developers to quickly fix bugs
- Makes schedule and/or scope slip obvious very early
Changing Requirements

- Optimize tests and testing process to respond to frequent change
- Treat this as part of the key contribution of this approach to development, so actively embrace change
- Enable team interactions that support positive responses to change → build trust!
  - Bug vs. feature vs. enhancement discussions
  - Rapid turnaround on bug fixes
  - Valuing what the customer wants vs. what is easy to implement

Short Timelines

- Requires rapid development and updating of tests as designs and requirements are finalized or changed
- For a 6 week increment, this means the following schedule:
  - Week 1: User Acceptance Test (UAT) for prior increment done and results reviewed with business owners
  - Weeks 2-4: Develop integration / “acceptance” tests for the increment, re-test bugs as they are fixed, run integration tests, report results, re-test….
  - Week 5: Get consensus on scope of functionality for next UAT; prepare user documents and send out
  - Week 6: Full regression plus remaining integration tests for the increment
Business users as TEAM

- Working together daily means more potential for conflict as well as cooperation
- Usual outlets for frustration not readily available
- Requires courage for all, especially the business user who may be isolated from their peers and normal work environment
- Can reduce the need to “sell” the system during implementation
- Requires more professionalism on a consistent basis
- Supports building communication around the business problem domain

Face-to-face conversations

- Reliance on verbal communication goes against the traditional grain and only works well for some kinds of things
- Client needs something for reference when they go to maintain the system after the project is done
- Can be challenging when things go wrong as memory is a horrible thing to rely on
- Requires stable team member composition since knowledge becomes embedded in individuals
Simplicity is essential

- Build the simplest tests and test framework that will work
- Include complexity where it is important or risky for the business and be able to justify it
- Take the time to “write it succinctly” - this is harder than being verbose!
- Resist the temptation to make it cool because you can

Test Design Process

- Key influences from the Agile internal process:
  - Heavy business involvement
  - Embrace change
  - Frequent, iterative releases of usable code
  - Desire for just-good-enough documentation and status reports
  - Focus on seeking the simplest possible solution
- Key influences from the Waterfall external environment:
  - Communication through project artifacts
  - Use of traditional-looking techniques
  - Use of BAs for “happy path” confirmatory tests
Techniques Selected

- Exploratory testing, based on Session-based testing methods from Bach & Bach
- Automated integration / regression tests (aka “acceptance” tests in Agile writings)
- User Acceptance Testing (UAT), very customized process

Exploratory tests

- Natural fit for requirements that evolve and are loosely documented
- Existing, well thought out approach available
- Works well with time-boxed releases
- More formalized version of existing client testing practices
  - Enables gentle mentoring
  - Missions can be different for client BAs and QA staff
  - Discussion of findings supports teaching through examples
  - Basics of process can be easily grasped, allowing client BAs to learn at their pace
Automated Integration / Regression Tests

- Good fit with automated unit tests, so is comfortable for development staff to accept
- Good support from open source tools for this style of testing
- Reduces fear of change both internally and externally through frequent, predictable testing

Drawbacks:
- Scripts do create documentation of tests, but may be too technical to suit the client business needs
- Fragility of recorded scripts for frequent changes (address through modular architecture and grey box test approach)
- Management of test cases not well supported (tools typically support too heavy or too shallow a structure)

User Acceptance Tests

First, a little background:
- Had first UAT session scheduled 2 days after I started
- Application had very limited functionality available and very basic user stories defined
- Vague goals, expectations, and undefined participants for the process
- BAs and developers had been given limited direct access to the users

With little time to define the process, there wasn’t an immediately obvious applicable example to follow, so…

Define it from scratch, working with what I had
UAT process

For the first session:
- Frame it as a chance to try out the general process
- Very limited functionality allowed us to focus on getting comfortable with the expectations
- State up front that the process will change over time as needed to be useful
- Set their focus through the user stories
- Introduce the development team to “personalize” both sides to each other
- Use active engagement techniques to encourage participation early and often
- Use group review session to establish consensus and isolate conflict among diverse users
- Use business owner review to allow a more considered evaluation of business priority to control scope creep

UAT Process Themes

- Use group discussions and business owner review sessions to keep project from becoming the middleman in existing business conflict
- Control the quality of the bug reports by having QA create and maintain the defect tracking system
- Set realistic expectations at each session for the probability of implementing any changes
- Provide detailed information on what was changed as a result of user feedback
- Actively elicit feedback on the process
- Provide just-frequent-enough interaction between the project team and users
- Solicit and pass along kudos
UAT Challenges

- Good facilitation skills required
- Decent requirements documents needed
- Developer willingness to code in chunks that match user stories is vital
- Time to test the release and get urgent fixes done prior to the UAT without going too long after the developers worked with the code
- Time to gather and filter UAT feedback
- Working out who “owns” the design decisions - dev, BA, QA, user, or business owner?
- Using terminology that may have a history for some project team members

UAT Gains

- UAT team became solid project advocates to the dispersed users
- Increased business owner confidence in the system
- “Built in” demo scripts for stakeholder presentations
- Increased business support in resolving outside project dependency problems (no need to separately educate them on the issues)
- Better cooperation and understanding between business and project team
- **Convergent view of perceived quality**
Lessons Learned

- Importance of Vocabulary for those with a history
- Function of “UAT” for the client as part of adoption
- Function of “UAT” for the team as vital feedback
- Need for education on new or adapted processes
- Need to adhere to external (PMO) requirements
- Baby steps are better than falling down the stairs
- If a requirement has a short life expectancy, it may be helpful not to create a false impression of permanence or precision

Lessons Learned

- Reality of supporting 9 developers with rapid turnaround on tests - it’s hard to keep up!
- Starting slower may be a good long run strategy (see Brian Marick’s comments on this approach http://www.exampler.com/blog/2007/04/28/going-slow/), especially if you are trying out new tools or processes
- If quality is evaluated by comparing observation to expectation, and both are subject to perceptual biases, then quality can be “changed” without changing a thing
Questions & Comments?