

2008

PACIFIC NW SOFTWARE
QUALITY
CONFERENCE



COLLABORATIVE
QUALITY

OCTOBER 13-15, 2008

CONFERENCE PAPER EXCERPT
FROM THE

CONFERENCE
PROCEEDINGS

Permission to copy, without fee, all or part of this material, except copyrighted material as noted, is granted provided that the copies are not made or distributed for commercial use.



Acceptance Testing: A Love Story in Two Acts

Grigori Melnik , Microsoft Corporation
Jon Bach, Quardev, Inc.

*For presentation at the 2008 Pacific Northwest Software Quality Conference in
October 2008*

Abstract

This report is about the experiences of five software professionals who set out to produce a book on acceptance testing for other software professionals around the world. This team, headquartered at Microsoft's Redmond campus in the *patterns & practices* group set out to collaborate not only with each other, but with other Microsoft internal teams and external reviewers. After research, we discovered two major “acts” or phases in play when preparing software for acceptance by a customer: the Readiness Assessment and the Acceptance Decision. This may seem obvious, but we found the existing literature and guidance in these areas to be sparse. This paper is about what we did to elucidate activities inherent in each of these phases.

Author Biographies

Dr. Grigori Melnik is a Senior Product Planner in the *patterns & practices* group at Microsoft, leading the Process & Engineering Practices focus area. Prior to that, Grigori was a researcher, software engineer, coach and educator with 15+ years of meaningful industrial and research experience. His areas of expertise include agile methods, empirical software engineering, software testing and test automation, and software economics. Grigori is a regular contributor and speaker to software engineering conferences and workshops around the world. Grigori was Program Chair of the Agile 2008 conference and a member of the IEEE Software Advisory Board.

Jon Bach is Manager for Corporate Intellect and lead consultant at Quardev, Inc – a Seattle outsource test laboratory that also provides technical writing, training, and consulting services in software development. In his 13 years’ experience in software testing, Jon is most famous as an exploratory testing expert and frequent speaker about the cognition and management involved in testing – most notably as co-inventor of Session-Based Test Management. Jon was also president of the 2007 Conference of the Association for Software Testing in Seattle and a recent recipient of Best Presentation at the 2008 Software Testing Analysis and Review (STAR East) conference.

Rationale

In 2007, the Microsoft patterns & practices team did some research into the existing body of knowledge on acceptance testing practices. We found a noticeable lack of consistency and actionable guidance. Furthermore, we found IEEE's definition limiting:

*"Formal testing conducted to determine whether or not a system satisfies its acceptance criteria and to enable the customer to determine whether or not to accept the system."*¹

The main problem with this definition was that it was recursive -- it used the word "accept" in its own definition without explaining what it means. So after some brainstorming, we came up with something we felt was more helpful:

"Planned evaluation by a customer (or customer proxy) to assess to what degree a system meets their expectations."

With help from a group of 10 Microsoft internal and external experts assembled into an advisory board, we had carefully crafted each word of the definition until it made sense to all of us. We took each word and decided to make a chapter out of it to explain exactly what we meant by our definition.

For example:

- **"Planned"** -- What does it mean to plan? Who does it? What considerations are there?
- **"Evaluation"** -- What's involved in evaluation? Who does it? What techniques can be used?
- **"By a customer"** -- Who is the customer? Are they different than the user? Should they be an expert in the domain?

- **“Or customer proxy”** -- Who is an acceptable customer substitute? How might their expectations vary from the customer?
- **“To Assess to What degree”** -- What are the methods to determine whether something is acceptable? Under what conditions might it be more or less acceptable? Is it possible for something to *almost* meet acceptance?
- **“A system”** -- Is it an application or a service? What comprises the system? What is being delivered? Will it change over time?
- **“Meets expectations”** -- What’s the difference between implicit and explicit expectations? Are requirements the same as expectations?

These questions not only framed our vision and scope, they provoked even more questions. Answering those questions became an anchoring mission for us because, like software testing, questions often drive the effort. It drove our planning, writing and research.

Our first bit of research (see Appendix) consisted of an anonymous online survey and resulted in data from 127 software professionals of varying titles and responsibilities. It helped us discover the following:

- Acceptance testing was not being addressed as an engineering discipline.
- There was no coherent story about how to plan and execute notions of “acceptance”.
- There were no specific “how-to's” centered on how to actually plan and execute acceptance testing activities.
- There was no guidance for knowing when you’re done with acceptance testing.
- There was no guidance around how to get a good customer (or customer proxy) to participate in providing acceptance tests.
- There were minimal testing checklists.
- There were lots of non-actionable items from IEEE standards, which tend to include recursive definitions.

- We found some research on acceptance testing, but results were not validated by the industry.
- Most engineering teams look at acceptance testing as a “necessary evil”.
- With existing notions of acceptance, tester and customer roles are not well-defined, leading us to be confused about who does what and what Functional Acceptance vs. Customer Acceptance means.

From this, we decided to create another non-scientific online survey to get more depth. It contained a mix of multiple choice, multi-select and open-ended questions like:

- *Choose the definition that is most closely aligned with your understanding of what -acceptance testing is.*
- *On your current team, who specifies the acceptance criteria for features/stories?*
- *On your current team, who specifies the acceptance criteria for product/service releases?*
- *How are acceptance criteria communicated to the development team?*
- *How would you prefer the acceptance criteria to be communicated to the development team?*
- *How do you know that you are done, i.e. the product is ready for release?*
- *What do you expect our guide on acceptance testing to have to be useful to you and your team?*
- *Which of the following types of testing should not be included in acceptance testing?*
- *Please share a typical acceptance test or acceptance criteria.*
- *In your opinion what is the biggest challenge with acceptance testing on your project?*

- *What is the level of customer involvement on your project?*
- *How much do you outsource or offshore part or all of your development, testing, or acceptance testing?*

From this feedback², we created 3 guiding missions:

- 1) Inject Acceptance Testing into all phases of a project.
- 2) Promote it as a discipline (not a series of tasks but a context-driven concept).
- 3) Promote the notion or “readiness testing” as well as acceptance.

Furthermore, we wanted to have how-to's and checklists – artifacts around planning and execution that were actionable and scalable, and we wanted to ground them to specific examples using a sample application.

Process

We took an Agile development approach, first creating a project wiki for team members to communicate, post content, and keep track of progress.

Here is a diagram of our process:

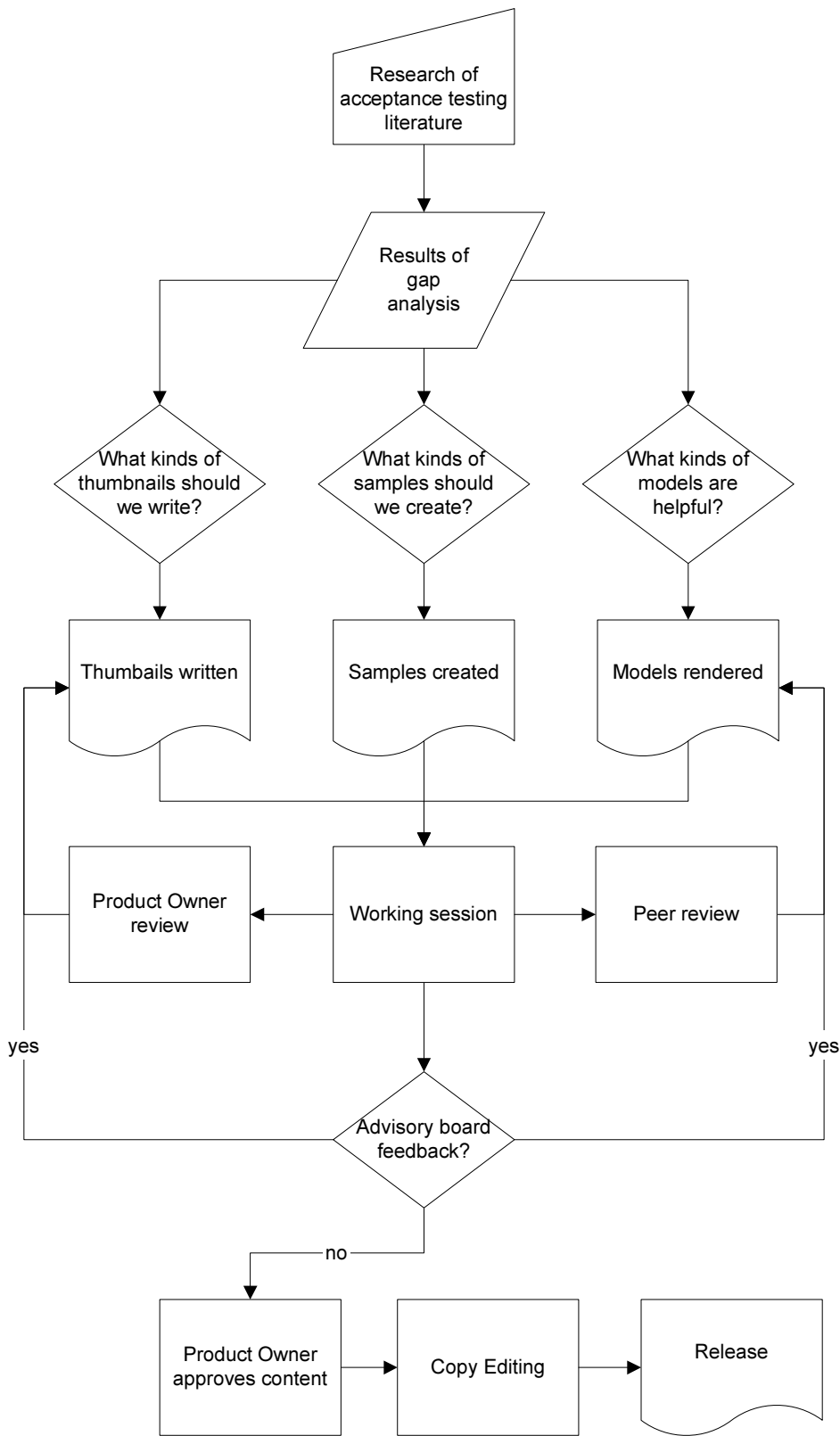


Figure 1: Dataflow diagram

What the diagram doesn't show very well is that we produced content in weekly iterations and had daily stand-up meetings where team members reported their progress for the last 24 hours.

Our story backlog was a list of questions we thought the readers would have about acceptance testing, which became post-it note tasks on a whiteboard. When at least 2 pages of content were created to answer each "story" (and when we could tell a good narrative about that topic), we considered that topic a "thumbnail sketch" and marked it as done, meaning it was ready for someone else on the team to review.

Content structure

The following is a list of some of the approximately 50 thumbnail sketches we produced that we wanted readers to consider in planning and doing their readiness and acceptance testing activities. They cover test processes, requirements, and test design practices, test management, test automation, functional and parafunctional testing, and test oracles:

Test Processes

Exploratory Testing – an approach to testing where testers are in control of the design and execution of their testing and use of techniques as they learn about the product. The point is to react to new or emerging information, so the tester is encouraged to change tactics to follow hunches and discover important issues.

Acceptance Test-Driven Development – a way to write software, starting with the customer requirements and the customer's specified acceptance criteria/tests for those requirements, and using them as the basis for all development.

Incremental Acceptance Testing – when the development of functionality is organized so that individual features can be acceptance tested as soon as they are deemed ready by the supplier team instead of waiting until the end of the project when all of the features are done.

Regression Testing – testing to ensure that perceived quality is not going down due to new features implemented or bugs fixed. Regression testing minimizes risk of new problems by running a standard set of tests on each release candidate.

Script-Driven Testing – preparing tests in written form well ahead of their execution.

Test-Last Acceptance – when acceptance testing is done at the same time as the decision -- after all development and readiness assessment activities have been completed.

Vision-Scope – a way to frame the objectives of the mission by meeting with stakeholders about the following 5 elements: Customer, Needs, Product, Value, and Purpose.

Risk Assessment – identifying things that could go wrong on the project and classify by likelihood and impact to help prioritize the risk mitigation activities including, but not restricted to, testing.

Requirements Practices

Requirements – explicit, stated expectations of a user or customer – the desires for functions that solve some kind of problem or set of problems – but they may also be implicit, assumed and unstated.

Use Case Modeling – a way to describe the functional requirements of a software-intensive system. It focuses on the goals of what the system’s users would like to achieve while using the system and what the system needs to do to help them achieve the goals.

User Stories – a way to manage highly-incremental development by deriving and documenting user actions and behaviors from requirements and establishing those as stories that serve as a unit of development work.

Test Design Practices

Regulatory Considerations – sometimes the customer or supplier is bound by forces that affect the way they approach acceptance testing. Government agencies have their own criteria, standards and rules that govern the way software can function.

Scenario Testing – unlike functional tests based on use cases, scenarios typically incorporate behavior from many use cases into the same test based on actual or possible usage behaviors. Scenarios are typically expressed in natural, ubiquitous language.

Soap Opera Testing – a test technique that get its name from fictional daytime television shows that have their roots in the 1950's and 60's when sponsors were often soap companies. An opera is an epic story, either a long series of events or a short series of very dramatic events happening to fictional characters.

Test Management

Cycle-Based Test Management – a method for managing testing effort where the test phases of the project are subdivided into two or more test cycles separated by periods of time set aside for bug-fixing.

Session-Based Test Management – a method for managing testing effort by compartmentalizing testing activity into time-boxes called sessions. It is most commonly used to manage exploratory testing but could be used for managing any kind of test execution.

Test Automation

Test Automation – an approach to testing that leverages the speed and power of computers to run tests that benefit from repetition or complex calculation.

Data-Driven Testing – a technique for reusing the same test logic on many sets of data values. The test is structured to read the input and corresponding expected output data values from a file or table and runs the same test logic with each set of data.

Hand-Scripted Test Automation – when automated test scripts are hand-coded in a scripting or programming language by people with enough technical skills to do some programming and debugging.

Keyword-Driven Testing – a technique for separating the specification of tests from the underlying mechanism to execute the tests by structuring test steps as action keywords followed by action-specific arguments.

Record-Refactoring – a common way to leverage the strengths of recorded tests without taking on the weaknesses involves refactoring. Refactoring is a way of re-organizing code script to remove duplication and make the code script simpler and easier to maintain without effecting affecting what it does.

Testing Functional Requirements

Ubiquitous Language – effective communication between business users of software and the technical builders and testers of software requires a common language. Since business people are not likely to learn technical jargon, the technical people must learn to speak “business”. This ubiquitous language should form the basis of all communication including the acceptance tests that describe what done looks like.

Workflow Testing – a test technique designed to verify how the system supports or implements a business process by executing a series of user actions toward a given task or objective. They often include tasks carried out by multiple users exercising different part of the system in a business workflow from a beginning state to an ending state.

Business Unit Testing – verifies the behavior of a business algorithm or business rule outside the normal context in which the algorithm or rule is utilized.

Combinatorial Testing – putting attributes of test criteria together to see if there are harmful interactions.

Installation Testing – once a software system is created, you need a way to get the components of the system deployed. Installation testing is ensuring that the deployment and removal of the components works.

Testing Parafunctional Requirements

Parafunctional Testing – testing that goes beyond confirming function, especially if the customer has implicit requirements, or requirements that go beyond a specific function, or expectations that emerged well after they were interviewed about requirements.

Usability Testing – a test technique designed to find out whether the product meets the needs of real users by watching users operate the product while trying to accomplish a specific task in a (nearly) realistic setting.

Security Testing – testing designed to reveal risks, vulnerabilities, attacks, and threats that would compromise valuable user or company data.

Performance Testing – a determination of whether or not the system under test meets the baselines with respect to throughput, bandwidth, or expected response times for user inputs.

Fuzz Testing – a way to test the robustness of an API or program input fields and entry points by feeding it random or pseudo-random data in many forms, usually in an automated manner.

Accessibility Testing – the act of preparing your software to be compliant with principles designed to help by people with varying degrees of capabilities.

Test Oracles

Comparable System Test Oracle – where the pass/fail status of a test is determined by comparing the actual results from the system under test with the result produced by a system with comparable functionality.

Hand-Crafted Test Oracle – when pass/fail status of a test is determined by comparing the actual results from the system under test with an expected result that was previously hand-crafted by a Human Test Oracle.

Human Test Oracle – when pass/fail status of a test is determined by a human subject matter expert inspecting the actual results from the system under test and deciding whether they are acceptable.

Previous Result (Regression) Test Oracle – when the pass/fail status of a test is determined by comparing the actual results from the system under test with the result saved when the same test case was run against the same system at some point in the past.

It's important to note that not all of these thumbnail ideas were created from the start. We let the list evolve and expand based on our research and feedback from the advisory panel; mostly because our best work came from 3-hour discussions we called "working sessions" – dedicated times when the team collaborated on something, whether it is a model, a strategy for production, or a research topic.

Ultimately, all new stories and ideas for content (thumbnails) were presented and approved by the content owner (Grigori Melnik) and project sponsor.

Table of Contents as project compass

Guiding the working sessions was our table of contents which we kept in an Excel spreadsheet. It evolved from a live TOC on the wiki page, then to a live

document in Microsoft Word, then to a live document on Sharepoint exposed through an extranet site so that the reviewer committee could see it.

Each row of the sheet was a thumbnail topic on which we tracked progress. There were 17 levels (columns) of “done-ness”, each level getting us closer to final release. This was our main vehicle to gauge whether or not we were still in readiness (ready for a team review) or ready for acceptance (product owner review and release to the web).

Below is a graphic which shows a snippet of a few thumbnail names and its level of done-ness.

Original #	New #	Thumbnail	Stowaway	Pilot Review	AD Review	Zed Done	Zed Free	ADRG	POB Actions	Narrative Done	Advisory	PDR Headings	PDR Fragments	CTP	Bicycle	Content Post	GPMAs	Copyedit	Approve	Sample
3		Thumbnail																		
46	7	7	Threat Modeling																	
47			Security Reviews																	
48			Security Test Planning																	
49	31	31	Fuzz Testing																	Fuzz Testing (GBS)
50	33	33	Performance Testing																	
51	34	34	Stress Testing MERGED WITH PERF TESTING	MERGED WITH PERF TESTING THUMBNAIL																
52	32	32	Usability Testing																	

Figure 1: Table of Contents: our main readiness / acceptance gauge for each artifact we produced.

Where did the ideas for thumbnail come from? We created a list of 120 questions we knew readers would likely ask, including:

- What are the common reference models?
- What are the kinds of acceptance testing?
- What is the lifecycle of an acceptance test?
- What are the possible phases?
- What is Readiness / Acceptance and how does it help me conceptualize AT?
- What are the existing standards for AT?
- What is the difference between Agile (incremental) and Waterfall?
- Who is going to do the testing?
- How do you select proxies?
- What are some questions to ask test outsourcers?
- What are the strategies for AT?
- What does an AT plan look like?
- How much detail is needed in the plan?

- Does it cover enough risks?
- How are the risks represented?
- How to get buy-in from stakeholders?
- What does a Vision Statement look like?
- What does a Scorecard look like?
- What gets tested?
- What kind of test bed should we build for data?
- What tools should be used?
- How does a maintenance contract affect acceptance?
- What kinds of customers might there be?
- How much testing do we need to (plan) to do?
- Who writes the acceptance tests?

From this, we looked for affinities. For example, was there a test technique or approach that would answer more than one question? If so, it might be worth writing a thumbnail about.

Samples

To ground our answers to these questions, we also created samples to illustrate the topics in our thumbnail sketches. Borrowing from Microsoft's Visual Studio Team System's example, we used a fictional software service called the Global Bank Identity Theft Protection System (ITPS).

The sample Vision/Scope for the ITPS feature read as follows:

"For current Global Bank premium account holders who need to monitor their accounts for suspicious activity like identity theft, fraud, and infiltration the Identity Theft Protection Service (ITPS) will allow customers to sign up for notification of suspect transactions by email, IM, text, and/or voice that provide general information and a URL for secure login to review transaction details unlike that for non-premium account holders (less than \$50,000 in assets) or premium account holders at other competing banks."

The samples we produced around this to help ground the content in our thumbnail sketches included:

- A sample vision-scope
- An all-pairs modeling sample

- A bug chart sample
- A risk assessment sample
- An exploratory session report
- A sample performance testing report
- A sample test plan
- A workflow example
- A soap opera test
- A scenario sample

It was our hope that creating samples like this would help readers understand why a thumbnail sketch of an n approach or a technique could be meaningful, either during readiness or acceptance.

Models

Other than questions, thumbnail sketches, and samples, we knew we had to create ways to illustrate our research and ideas to have a strong visual component to the book. Models served as a way to represent guiding principles for us in our work, that sometimes framed discussions for specific topics.

For example, from one working session came the idea for a working model about how decisions get made:

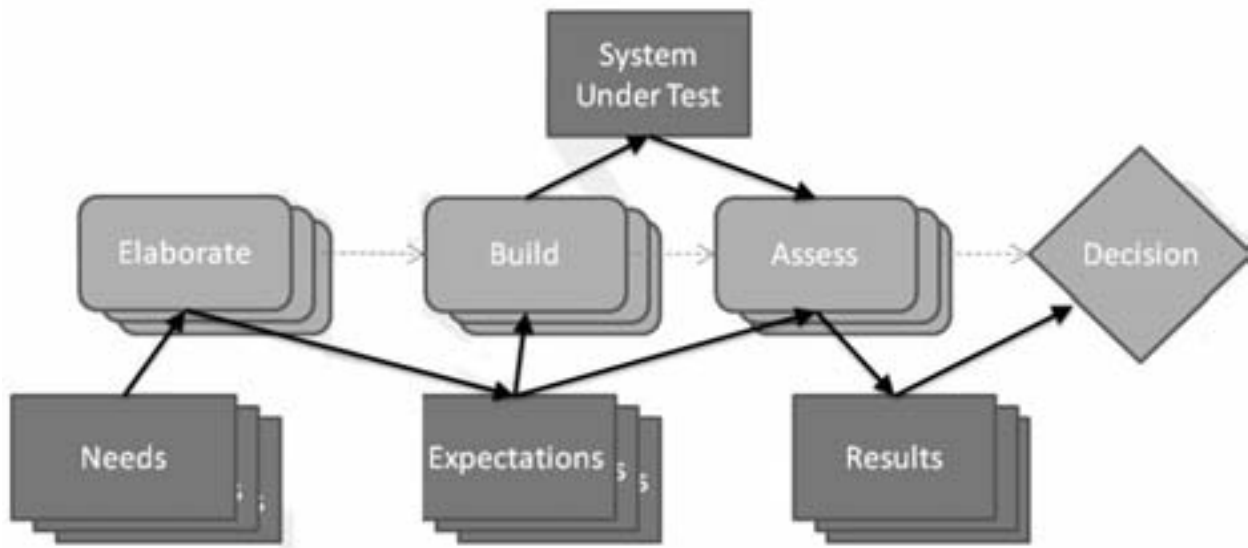


Figure 2: Decision-Making Model

The diamond on the right represents the decision which is made based on the test results. The test results are based on the testing / assessment activity which assesses the system-under-test against the expectations. The expectations of the system-under-test were defined based on the needs / requirements of the users. While many of our thumbnail sketches describe how to do the assessment activity, others describe ways to define the expectations based on the needs. That is one of the reasons our guide has a number of requirements-related practices; it's not about testing, it's about acceptance.

But it was only when expanding on this did in another working session did we get the biggest epiphany. It had to do with a simple question: How would we apply this decision-making concept to our own project?

Our answer took the form of what we now call the Gating Model.

(Notice that the figure below is a hand drawn sketch which the authors of this paper decided to leave as-is to talk about the value of low-fidelity prototyping we used on this project. That is, sometimes a hand-written model can be just as effective as one produced with Publisher or Excel when explaining something.)

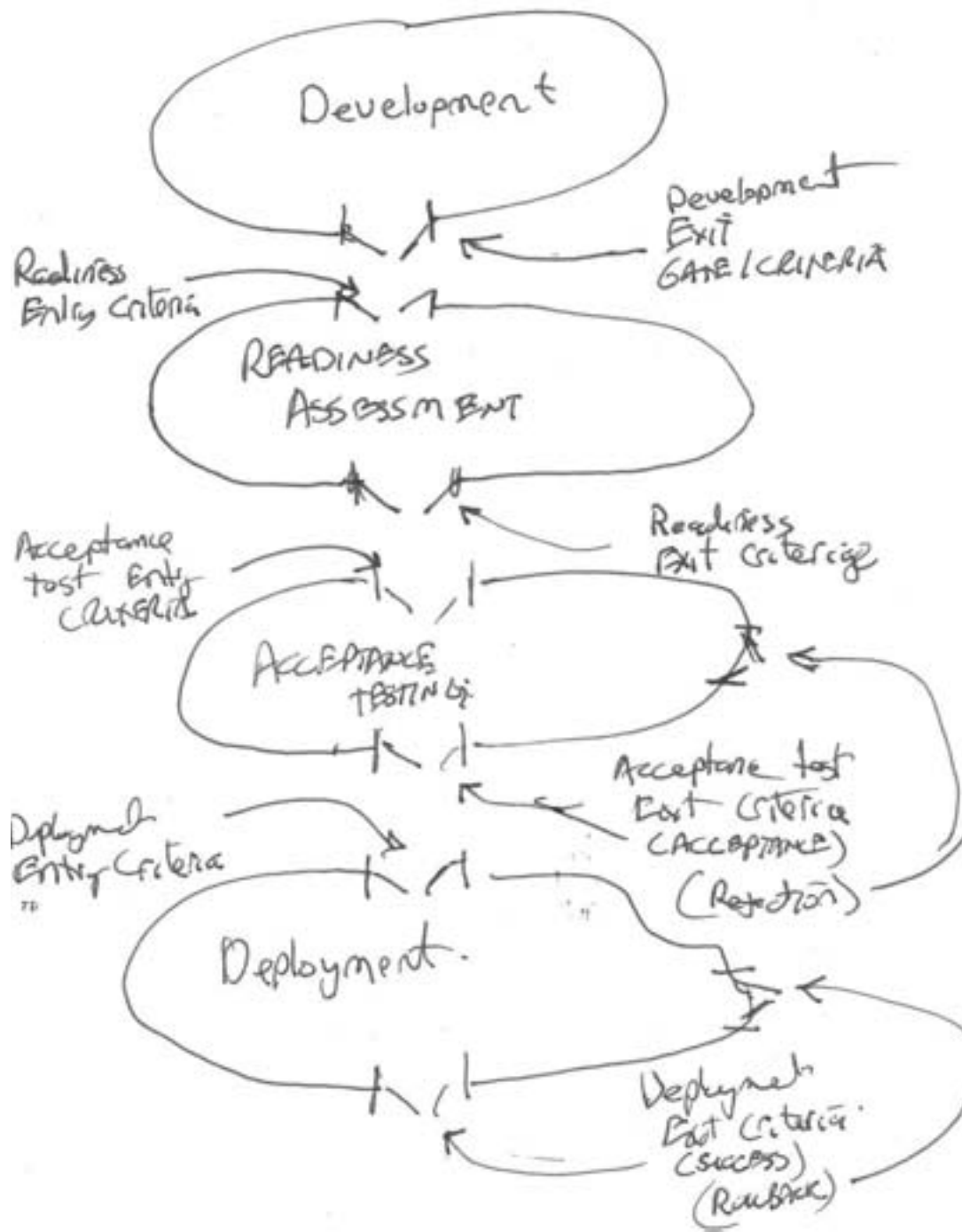


Figure 3: Gating Model

Readiness Assessment Phase

Readiness assessment is done by the supplier of the software before declaring the system "ready for acceptance testing". As a result, the gate between

Readiness and Acceptance Testing phases is largely staffed by gatekeepers belonging to the supplier.

For the supplier to feel confident that their product will pass muster with the customer, it may require that they do a lot more testing than the customer might do during acceptance testing. As a result, the testing done as part of Readiness assessment is likely to be much more exhaustive and rigorous, and employ a much wider array of testing techniques than that done during the actual acceptance testing.

Acceptance Testing Phase

The acceptance testing phase is when the customer (or their proxy) is actually executing tests that will help them make the decision to accept or not accept the software. The main entry criteria for the acceptance phase are that the supplier has deemed the software ready for acceptance testing. Secondary criteria may include whether or not the customer is sufficiently prepared to conduct the acceptance testing.

Exit criteria are primarily focussed on whether or not the “accept software” decision has been made. The software is considered in acceptance testing until either

- The customer has accepted it

Or

- The customer has found it so inadequate as to reject it outright. At this point, the ball is back in the supplier’s court until they have done further development and readiness assessment based on their revised understanding of the customer’s expectations.

Challenges

Like most development projects, our effort to produce a book on acceptance testing guidance was not without its challenges.

First, not everyone on the team was a dedicated full-time resource. Other projects competed for our attention and made progress slow at times. The dedication problem was the same for our advisory board, as most of the

people who signed up to be on it had other commitments that kept them from helping us flush out important errors and omissions in our content.

The presence of the advisory board made it necessary to abandon the wiki pages we had started at the beginning of the project because it was an internal resource and reviewers could not get access to it. We adopted SharePoint as a hosted content management system for our extranet site.

We also could not identify our velocity -- delivery of some unit over a given time period -- as can be done on most projects that use Agile methods based on a unit of work like story cards or code production. The classic way of doing velocity in Agile is to put an estimate on every piece of content that is meant to pass acceptance and then measure how much of that got done per iteration. There was a time when we did not have the concept of a thumbnail, so it was hard to know what "done" looked like. Even when we had the concept of the thumbnail, they were not equal units of work -- some topics took longer to research and write than others. We consciously decided not to estimate how long it would take to complete a given thumbnail topic, but in retrospect, we could have used a first order approximation in the spreadsheet (e.g. squares in the Table of Contents spreadsheet). We also had a tough time predicting what would get done in a given time frame because we did not have the notion of a burn-down chart.

Another challenge was keeping our focus on acceptance testing. The more we researched and produced artifacts about testing, the more the book started to feel like a book on general testing techniques and approaches, not just about how those techniques could be used for acceptance. Creating a template for the thumbnails helped us stay focused and consistent, even though there were several content contributors on the project.

Summary

Based on a new definition of acceptance testing after surveying people in the industry, we used an Agile-style iterative and incremental development process. We strived for a continuous stream of value that started with a backlog of questions which served as "stories" about aspects of acceptance testing we wanted to write about. We used a live table of contents to guide our thumbnail sketches for content, to serve as project status reporting, and for

planning future iterations. In addition, we hosted weekly meetings with an advisory board consisting of people in software development who served as our first consumers of our guidance. We used a sample application to produce grounded artifacts to our thumbnails. We held working sessions, which helped us create models and new topics to go into the list of artifacts we thought would be helpful.

We found acceptance testing to be more than just creating a checklist of tests for the customer to run. The more we realized how thoughtful we were trying to be while producing a meaningful book on acceptance testing, the more we saw a parallel between producing software and producing a book – that is, sometimes you have to use a variety of techniques to anticipate how a customer will formally sign-off on their acceptance.

We found the relationship between Readiness (supplier-focused development) and Acceptance (customer-focused testing) to be a reinforcing synergy – a love story, if you will – which frames many kinds of test approaches and techniques to increase the likelihood that customers get their requirements, expectations, and needs met.

References

1 (* Source: SEI/CMU and Institute of Electrical and Electronics Engineers. IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries. New York, NY: 1990.)

2 See Appendix for survey

Appendix

This is a public survey we created and posted online at www.codeplex.com at the beginning of the project to get a sense of how “acceptance” was being performed on a variety of software projects. The objective was to form a foundation of acceptance testing and focus on strategies of readiness and acceptance.

Acceptance Testing Survey

Jun 16, 2008 11:18 AM PST

1. Choose one the following definitions that are most closely aligned with your understanding of what acceptance testing is		
Formal testing conducted to determine whether or not a system satisfies its acceptance criteria and to enable the customer to determine whether or not to accept the system.	52	50%
Planned evaluation of a system by a customer (and / or customer proxy) to assess to what degree it satisfies their expectations.	19	18%
Formal test of a system performed after installation on the customer's premises by an authorized entity with the participation of the supplier to determine whether the contractual obligations are met.	12	11%
Evaluation of a system by a technical tester as a way to transfer ownership of the application from the developers.	5	5%
Verification that requirements will always be met and will be acceptable by all users.	13	12%
Other, please specify	4	4%
Total	105	100%

2. On your current team, who specifies the acceptance criteria for individual features/stories? [Select all that apply]		
Customer	52	50%
Customer proxy	36	35%
Business analyst	34	33%
Program/Project manager	42	40%
Test lead/QA manager	29	28%
Tester	28	27%
Developer	18	17%
Development Lead	16	15%
Architect	12	12%
Release manager	4	4%
Compliance manager	5	5%
Usability specialist	8	8%
External consultant	0	0%
End-users	11	11%
CTO	2	2%
CFO	0	0%
CIO	1	1%
COO	0	0%
Legal counselor	1	1%
Other, please specify	5	5%

3. How are acceptance criteria communicated to/documented for the development team? [choose all that apply]		
Requirements Spec	39	38%
Statement of Work (SOW)	10	10%
Acceptance Document	16	15%
Use cases	25	24%
User stories	28	27%
Scenarios	17	16%
Legal contract/SLA	3	3%
Test plan	26	25%
Design documentation	18	17%
Manual test cases	22	21%
Automated test cases	18	17%
Wireframe	3	3%
Storyboard	8	8%
Face-to-face conversations	24	23%

Email/IM	13	12%
IEEE/FDA/ISO/DOD/SEI or some other standard or regulation	2	2%
Not at all	6	6%
Other, please specify	4	4%

4. How would you prefer the acceptance criteria to be communicated/documented? [choose your top three]		
Requirements Spec	43	41%
Statement of Work (SOW)	6	6%
Acceptance Document	33	31%
Use cases	29	28%
User stories	36	34%
Scenarios	26	25%
Legal contract/SLA	2	2%
Test plan	36	34%
Design documentation	13	12%
Manual test cases	14	13%
Automated test cases	29	28%
Wireframe	1	1%
Storyboard	9	9%
Face-to-face conversations	14	13%
Email/IM	1	1%
IEEE/FDA/ISO/DOD/SEI or some other standard or regulation	2	2%
Not at all	0	0%
Other, please specify	4	4%

5. How do you, in your role, know that you are done, i.e. the product, application, or service is ready for release? [choose all that apply]		
When customers realize enough value	15	15%
When user stories/requirements are complete	40	39%
Development task is completed, tested, demoed to and approved by customer	60	58%
Scorecard criteria are met	6	6%
Product has no critical problems	33	32%
Product has sufficient benefits	11	11%
The value of the benefits in the product outweigh the problems	10	10%
Impact of the last code modification is minimum and does not require full test pass	3	3%
Last test pass resulted in bugs that were relatively low severity and in low priority features	16	16%
Overall trend of bugs found in test passes indicates that code is becoming more and more stable	12	12%
Test coverage of the last successful test pass is acceptable	23	22%
All functional and non functional quality gates identified for the release have been met	38	37%
All deferred active bugs have been triaged / documented as known issues and shared with customer	28	27%
Regression test pass doesn't result in reactivation of fixed bugs	27	26%
Documentation is written/updated and edited	19	18%
Risks have been identified and mitigation strategies formulated	18	17%
Further time spent testing, designing, building is deemed more expensive (harmful) than helpful	12	12%

When the checklist is completed	16	16%
When there is no more time	17	17%
When there is no remaining budget	8	8%
Other, please specify	4	4%

6. How do you address bugs found during acceptance testing?		
Discard the current release candidate and request a new one with the bugs fixed	57	56%
Defer the bugs and address them as part of the service level agreement	13	13%
File a change request	19	19%
Other, please specify	13	13%
Total	102	100%

7. How would your acceptance testers define a bug?		
Non-conformance to the written specification or contract (SOW)	8	8%
Does not meet an explicit, written requirement in a requirements doc	24	24%
Does not meet an expectation (implicit requirement)	25	25%
Does not adhere to a story card	12	12%
Anything the customer isn't happy with when they see the software	27	26%
Other, please specify	6	6%
Total	102	100%

8. At what phase of the project does your acceptance tester get involved?		
During project planning	17	17%
During requirements gathering	20	19%
During system design	7	7%
When development starts	15	15%
When development is finished	28	27%
When the customer sees it for the first time	7	7%
Other, please specify	9	9%
Total	103	100%

9. From the perspective of defining requirements and performing acceptance testing, which of the following applies?		
Customer performs both	19	18%
Customer and in house test team perform both	34	33%
Customer defines requirements only, and in house test team does acceptance testing	26	25%
Customer defines requirements and outsources acceptance testing to an external vendor	2	2%
Customer defines requirements and end users perform acceptance testing	16	16%
Other, please specify	6	6%
Total	103	100%

10. What tools do you use for acceptance testing? [Select all that apply]		
None	43	43%

Commercial tools (Visual Studio Team System, Rational suite, Seapine products, etc.), please specify below	42	42%
Open source tools (Fit/Fitness/Selenium or similar), please specify below	17	17%
In house tools	21	21%
Other, please specify _____	1	1%
Specify:	8	8%

11. What do you expect our guide to contain to be useful to your team? [choose three]		
Descriptions of the various acceptance testing practices/techniques?	82	80%
Case studies/stories from the trenches	40	39%
Checklists	50	49%
How-to's	44	43%
Examples : test case/test strategy/test plan/test script	70	68%
Sample reports	18	17%
Metrics and how to use them	35	34%
Tips and Tricks / Heuristics	40	39%
Anti-patterns and known mistakes	59	57%
Other, please specify:	6	6%

12. Please share a typical acceptance test or acceptance criteria. [Do not worry about giving the context, we would like to simply see the style and structure of your acceptance tests]
56 Responses

13. In your opinion what is the biggest challenge with acceptance testing on your current/last project?
69 Responses

General Questions

14. Which best describes the process your team follows?		
Agile (Scrum, Crystal, Lean, Extreme Programming, and so on)	54	55%
Formal (Tayloristic, waterfall, spiral, and so on)	30	30%
Other, please specify	15	15%
Total	99	100%

15. What is (are) your role(s) on the team? [Choose all that apply]		
Customer	1	1%
Customer proxy	9	9%
Business analyst	7	7%
Program/Project manager	24	23%
Test lead/QA manager	18	17%
Tester	19	18%
Developer	32	31%
Development Lead	37	36%
Architect	30	29%
Release manager	9	9%

Compliance manager	0	0%
Usability/User experience designer/specialist	6	6%
External consultant	8	8%
Technical writer	5	5%
End-user	1	1%
Executive (CTO, CFO, CIO, COO and so on)	6	6%
Legal counselor	1	1%
Ethnographer	1	1%
Other, please specify	2	2%

16. What is the level of customer involvement/commitment on your project?		
Very High (on-site customer)	18	18%
High (near site customer or customer proxy)	28	28%
Moderate (customer present at weekly status updates)	22	22%
Low (customer gets only involved once a month)	15	15%
Very Low (customer gets only involved at the beginning of the project and at the final demo)	11	11%
What customer?	4	4%
Other, please specify	3	3%
Total	101	100%

17. How many people are on your team [again, please think of the most recent project]?	
98 Responses	

18. What percentage of your development and testing effort does your team outsource?	
96 Responses	

19. If you have a story or a lesson learned about acceptance testing that you'd like to share, feel free to write it below or provide an email address if you'd like to be contacted and interviewed.	
26 Responses	