Is Quality Negotiable?

Lisa Crispin
Senior Consultant
BoldTech Systems
1050 17th St. 11th Floor
Denver, CO 80204
USA
303.629.9206 x 5274
lisa.crispin@att.net

Abstract

The morning I sat down to start writing this paper, my contractor called (we're in the middle of building an addition to our house). He told me the painter would apply one coat of paint to the primed siding. If I wanted a second coat of paint, it would cost $275 extra. Higher quality often costs extra. It struck me how often we make decisions and compromises about quality in our daily lives. Shall I buy a Yugo or a Volvo? Eat at McDonald's or go home and cook? It all depends on what I need most – money, safety, time, nutrition.

In eXtreme Programming Explained [1], Kent Beck describes the four variables of software development: Cost, Time, Quality and Scope. As he says, "quality is a strange variable". If you try to save time or money, or increase scope, by sacrificing quality, you will pay a price in human, business and technical costs. XP teams have the right to do their best work.

On the other hand, customers have the right to specify and pay for the only the quality they need. How does one reconcile two potentially conflicting points of view? Is quality negotiable? If so, how do we go about negotiating it?

This paper will explore the following questions:

- Is quality negotiable?
- How can we negotiate quality?
- What are internal and external quality, and are either or both negotiable?
- What's the XP tester's quality assurance role?
- How far should testers go in helping the customer define acceptance criteria?

Keywords

Testing, acceptance testing, quality assurance, tester, web testing, GUI testing, XP, painting houses.

Introduction

When my husband and I decided to put an addition on our house, we chose to include a basement. We signed a detailed contract with our contractor which specified many little details. We thought we read this carefully. When the basement was built and a hole cut for the door, the contractor pointed out that he had neglected to include the door itself in the contract. We had access to the new basement – it was functional – just no way to close it off if we wanted. Since the door was not included, we would either have to do without it, or pay extra.

Naturally, we had *assumed* there would be a door to the basement room which we could open and shut. But since we had not specified this, the contractor hadn't included the price of the door or the labor to install it in his price. We couldn't expect the contractor to just give us a free door. How nice it would have been if someone else had looked at the contract with me and asked, "There isn't a door specified here, don't you want one?" Then I could have decided whether or not to spend the money – it wouldn't have been a surprise later.

I've participated in XP projects where I've seen this type of thing happen. (OK, it happens in all software projects, no matter what practices are

used). For example, the customer has a story for an add screen, and just assumes the developers know he also wants the ability to update, read and delete. Or maybe there's a story for a login screen with authentication, but nothing about what should happen if the same user logs in twice. At the end of the iteration, an exception thrown by having the same user log in twice looks like a defect.

As a tester and quality assurance engineer of long experience, I'm something of a tyrant about quality. I have my own standards which naturally I think everyone should follow. When I started working on XP projects, I realized it wasn't about MY quality standards – it was the customers'.

Here's an example. Say we have a startup company as our customer. For now, they just need their system up and running to show to potential investors. They just need a system that's available one or two hours a day for demos. They aren't looking for a bulletproof 24x7 production server. In fact, they can't afford to PAY for a bulletproof system right now. They'd rather have more features to show off, even if they might not handle a high level of throughput. It would probably take significantly more time and /or resources to produce a system with guaranteed stability. If the customer isn't willing to pay the price, they can't expect to get it for free.

In XP, the customer's role is to make business decisions, not to be a quality expert. Face it, some people are always on the "happy path"... just as my husband and I were when we signed a contract with our builder for our home addition.

As the tester, I feel it's my responsibility to help the customer make conscious decisions about quality during the planning process. If the customer is clear about his acceptance criteria, and these are reflected accurately in the acceptance tests, we're much more likely to achieve the level of quality the customer wants, without giving our time away for free.

Internal and External Quality

In Extreme Programming Explained, Kent Beck writes:

> *"There is a strange relationship between internal and external quality. External quality is quality as measured by the customer. Internal quality is quality as measured by the programmers."*

He goes on to explain the human effect on quality:

> *"If you deliberately downgrade quality, your team might go faster at first, but soon the demoralization of producing crap will overwhelm any gains you temporarily made from not testing, or not reviewing, or not sticking to standards."*

In this light, it looks as if we should always strive for the highest standard of quality. This would of course make me very happy. But is the customer willing to pay for it?

I think the important concept here is the difference between internal and external quality. Whenever I meet someone who works in an XP environment, they always tell me that one of the reasons they love coming to work each morning is they know they'll be allowed to do their best work. If you take that away, XP won't work. It's good to have 100% successful unit tests. In the long run, it speeds up development time. Internal quality should be a given.

External quality can be defined as a set of features, for example:

- Whenever the user makes a mistake, a user-friendly error screen appears
- It's impossible to crash the server via the user interface
- The system can handle a hundred concurrent logins
- The system will stay up 99.995% of the time

Negotiating with the customer on external quality doesn't mean skimping on acceptance tests or deliberately producing unstable code. It means that the customer asks for a certain standard of quality and pays for it. If they want a system to handle all exceptions, that should be in the story – or multiple stories. Story one says to implement this functionality; story two says to make the functionality work with N concurrent users hammering it.

The XP Tester as Quality Assurance Engineer

The XP books say that the customer writes the test. In Extreme Programming Explained, Kent Beck says customers need to ask themselves, "What would have to be checked before I would be confident this story was done?" This very

question implies tests that check for intended functionality, or what my boss calls "Happy Path" testing.

Beck goes on to say that XP teams should have a dedicated tester who "uses the customer-inspired tests as the starting point for variations that are likely to break the software." This implies that the tester SHOULD guide the customer in defining tests that will really stress the application. He also mentions "stress" and "monkey" tests designed to zero in on unpredictable results.

In practice, when I have neglected to negotiate quality with a customer, acceptance testing became as treacherous as the mud pit which currently surrounds the new wing of my house. I wrote and performed acceptance tests according to my own standard of quality. Naturally, the tests, particularly the load tests and "monkey" tests, uncovered issues. To the XP-naive customer, these just look like bugs, and they're upsetting. The customer starts to worry that his stories aren't really being completed.

The XP way to deal with any kind of issues or defects is to turn them into stories, estimate them, and let the customer choose them for subsequent iterations. We know we'll always have some defects and unexpected issues crop up. However, to minimize the pain of dealing with these, it's best to set the criteria for quality at the start of each iteration.

Set the Quality Criteria

As the XP tester, ask lots of probing questions during the planning game. If the customer says "I want a security model so that members of different groups have access to different feature sets", ask: "Do you want error handling? Can the same user be logged in multiple times? How many concurrent logins should the system support?" This may lead to multiple stories, which will make estimation much easier.

Our customers have rarely thought of things like throughput capacity and stability up front – rather, they assume that their intentions are obvious: "Well, of COURSE I want to have more than one user log in at a time". The tester should turn assumptions into questions and answers. This way you don't end up with doorless rooms.

Write acceptance tests which prove not only the intended functionality, but the desired level of quality. Discuss issues such as these with the customer:

- What happens if the end user tries a totally bizarre path through the system?
- What are ways someone might try to hack past the security?
- What are the load and performance criteria?

As a result of these discussions, you may need to get the team back together to see if stories need to be split up or new stories written, and re-estimate stories to reflect the quality standards the customer has set in the acceptance tests. The customer will have to drop a story or change the mix, but they will be happier with the end result. Higher external quality means more time and/or more cost! Both a VW Beetle and a Hummer will get you to the grocery store, but if you need to cross the Kuwaiti desert, you're going to have to pay for the vehicle that's designed for the job.

Participate in developers' task assignment and estimation sessions. Testers often have more experience dealing with customers and a better understanding of what the customer meant to request. If the story is for a screen to add a record to the database, it's likely that the customer also meant they wanted to be able to read, update and delete records as well. Get everyone back together if there have been assumptions or a disconnect in understanding. Testers are in a unique position to facilitate this process.

I work in the same room as the developers, pair with them when needed, participate in the standup meetings. At the same time, I try to have as much contact with the customer as possible: we discuss the tests, get together to run them, look at the results. Testers are part of the development team – much more so than in a traditional software process. But as a tester, you need a level of detachment; you have to be able to be an advocate for the customer and at the same time a guardian of the developers. This can be a lonely and difficult role at times. The beauty of XP is that you're never really alone. With the help of your team, you can enhance the customer's satisfaction.

Running Acceptance Tests

The fast pace of XP iterations makes it difficult for acceptance testing to keep pace with

development. It's much better to do the acceptance testing in the same iteration with the corresponding stories. If you've ever done "downstream" testing where you don't get the code until development is "finished", you know that developers are looking ahead to the next set of tasks. It's painful to have to stop the fun new stuff you're doing and go back to fix something you've already put out of your mind.

In our projects, the developers try to organize tasks so that they can give me components to test early in the iteration. This way I can find defects and they can fix them BEFORE the end of the iteration. This means that the estimates have to include time to find and fix bugs. I think it makes everyone happier. There will most likely still be some defects or issues left over that have to become stories for future iterations, but it's possible to minimize these, and we should try.

As iterations roll along, regression testing of acceptance tests from previous iterations also have to be performed. In an email to the YahooGroup extremeprogramming, Ron Jeffries [2] says that once an acceptance test passes, it should pass forever after, so any regression defects for previously working tests must be addressed.

How do you do acceptance testing that fast? That's another paper in itself, but here are some tips.

- Make acceptance tests granular enough to show the project's true progress. Fifty tests of ten steps each is better than ten tests of fifty steps each.
- Separate test data from actions in the test cases. Spreadsheet formats work well; we've experimented successfully with XML formats too. It's easy to produce scripts to go from one format to another; a script that turns your spreadsheet test data into a form your test tool can use is invaluable.
- Identify areas of high business value and critical functionality. Automate tests for basic user scenarios that cover these areas. Add to them as time allows – don't forget to budget time to maintain and refactor automated tests.
- Modularize automated tests; avoid duplicate code and create reusable modules. For example, if you are testing a web application, have a main script that calls modules to do the work of verifying the various interfaces such as logging in, running queries, adding records and so on. Split functions such as verifying that a given set of links is present in the HTTP response out into separate modules that can be reused from test to test and project to project.
- Make automated tests self-verifying. Both manual and automated tests should produce visual reports which tell "pass" or "fail" at a glance. One way to do this is to write test results out in XML format and have your team write a tool that reads the XML and produces an HTML page with graphic representation of tests passed, failed and not run.
- Verify the minimum success criteria. As they say in the Air Force, if the minimum wasn't good enough, it wouldn't be the minimum.
- Apply XP practices to test automation. Do the simplest thing that works, continually refactor, pair test, verify critical functionality with a bare-bones "smoke" test.

Conclusion: Delivering Quality

In the abstract, I asked some questions that I've discussed in this paper. Here's a summary of what I have concluded. Disclaimer: despite all most a year doing XP, I have almost as many questions as I have answers. Practice XP and come up with your own conclusions!

Is quality negotiable? If negotiation means a dialog between the tester and the customer to make sure that the customer has clearly defined his quality criteria and that the acceptance tests are written to reflect these, then quality is negotiable. Because you, as the tester, and the customer talk about all aspects of quality, the customer can be specific about what he wanted and perhaps even defined stories that address criteria such as stability and performance under load. The developers can accurately estimate stories, and the customer can get the quality he's paying for.

How can we negotiate quality? By asking lots of questions of both customers and developers and making sure that nobody makes assumptions. By making sure the customer understands how XP works and what his role is in the planning game and knows what to expect each iteration. By putting a price on quality in

the form of story estimates and letting the customer decide what is most important for his business.

What are internal and external quality, and are either or both negotiable? Internal quality is quality as measured by the programmers. XP works best when each member of the team is allowed to do his best work. Internal quality may actually save money. External quality is quality as measured by the customer. The customer has to pay whatever it costs, so the customer should set the standard. The XP team helps the customer do this by telling them how much the criteria for quality will cost, in the form of story estimates.

What is the XP Tester's Quality Assurance Role? Help the customer set quality criteria and write tests that verify them. Provide a reality check for the developers. Mentor the developers in testing and quality assurance practices. Developing and testing share a lot of skills, but are distinctly different professions.

How far should testers go in helping the customer define acceptance criteria? As far as possible in the given timeframe. Ask all the questions you can think of.

The XP books state that acceptance tests don't have to pass 100%. The closer you come to clearly and completely defining the stories and the criteria for proving the stories work, the closer to 100% success you will have.

Negotiating quality makes the end of each iteration much more constructive and comfortable. The customer is satisfied that the stories were completed. He knew what to expect and his criteria for quality were met. The developers are satisfied that they did their best work and produced functioning code that is up to the customer's standards. The tester is satisfied that the customer got what he paid for, without the developers having to give away the store for free.

The acceptance test results may prompt the customer to change his mind about what his quality standards are. That's OK, this is XP. He's allowed to reduce scope in return for increased quality. We'll negotiate about that in the next iteration.

References

1. Beck, Kent. Extreme Programming Explained: Embrace Change. Addison-Wesley, 2000.
2. Jeffries, Ron. Email "New file uploaded", on extremeprogramming egroup, March 12, 2001.

Lisa Crispin

Since July 2000, Lisa Crispin has enjoyed the challenge of working as a tester on teams using Extreme Programming (XP) practices. She currently is a Senior Consultant with BoldTech Systems (http://www.boldtech.com), a technology consulting firm. In her pre-XP days, Lisa was Quality Boss for TRIP.com, a travel website. She has also worked as a quality assurance and test engineer for more traditional software companies, testing database, client-server and 4GL software on a wide variety of platforms.

Lisa has presented or co-authored papers on quality assurance and testing at local, national and international quality and XP conferences, including Quality Week, Quality Week Europe, XP2001, XP Universe and PSQT. Her article “Extreme Rules of the Road: How an XP Tester Can Steer the Project Toward Success” was published in the July/August issue of STQE Magazine.