Carefree Highway: How an XP Tester Can Drive Success

Lisa Crispin
Senior Quality Engineer
Tensegrent

Like many other test engineers and quality assurance managers, I've experienced much frustration over the years with projects that are late, over budget and in the end don't meet the customer's needs. A 1995 Standish Group study showed that only 16.2% of software projects are completed on time and within budget. Why? Why? Why? I thought maybe if we just had a little more process, or more discipline in following it... but things only got worse as we moved into Internet time.

Reading Kent Beck's eXtreme Programming Explained was my epiphany. It reminded me of my days as a baby programmer, where instead of writing reams of specs, we prototyped everything and reworked it in short iterations with our customers sitting alongside. That had worked pretty well, but ever since, I had been in traditional waterfall methodologies with varying degrees of success. Finally, here was a set of practices which perfectly fit my own style. See Sidebar 1 for a list of XP practices. Brian, I agree, p. 54, do you need me to type it up?

Driving the XP Car

Kent Beck compares XP to driving a car: you have to watch the road and make continual corrections to stay on track. When I started working as the XP tester on a team of up to 10 developers, I wondered: Who is driving this car, and how can I help make sure it arrives safely at its destination? What keeps the team from getting lost or making a wrong turn? Who makes them *STOP AND ASK DIRECTIONS*? You guessed it: the tester. The tester acts as the navigator, reading the acceptance test "maps", interpreting the customer's requests, watching for signposts in the form of acceptance test results, letting everyone know how the journey is progressing. The tester wears other hats too: customer advocate, developer guardian ; part of the XP team, but with a level of independent objectivity.

Through many months of trial and error, I have discovered major contributions that a tester can make in helping the XP team arrive safely and on time at its destination. Although I use the term "tester" here to refer to the position you fill on the XP team, you will use both testing and quality assurance skills. Let's follow through a typical iteration, say, two weeks, of an XP project to examine the roles of an XP tester. During this ieration, the developers will fully implement stories chosen by the customer, and acceptance tests will prove to the customer that the requested functionality was delivered. (*this sounds awkward but it's hard to define in a few words*

First Gear: Planning Game

We begin each iteration with a Planning Game. The customer writes up "stories", basic

descriptions of features she wants to have in the system. The developers estimate the cost of each story, and tell the customer what their budget is for the iteration. We use non-denominational points called "nudies". Let's say we finished 75 nudies in the last iteration; this iteration, the customer gets to choose 75 nudies' worth of stories.

As the tester, you ask lots of questions during the planning game. If the customer says "I want a security model so that members of different groups have different capabilities", ask him: "How should the error handling work? Can the same user be logged in multiple times? How many concurrent logins should the system be able to handle"? In our XP projects, I've found that the customer rarely thinks of things like system load capability and stability, but rather makes lots of assumptions. Assumptions are sharp tacks on the XP pavement; they're almost impossible to avoid. Developers are used to being isolated from the customer and having to make assumptions. Customers just expect that their intentions are obvious ("well, of COURSE I want to be able to have more than one user in the system at a time.) Your job is to turn assumptions into questions and answers before a tire blows.

Stop, Look, Listen...

Once the customer has selected the stories, it's time to break them into tasks and estimate each task - again, we'll use our non-denominational point system. We write the tasks on a whiteboard along with the estimates; team members get to take turns choosing tasks for which they'll be responsible. Listen to the developers as they list and estimate the tasks. They make assumptions too. I've occasionally discovered that the developers have all heard something different from the customer than what I heard. For example, we had a story from the customer that said "screen to create a customer record". The customer ASSUMED that we understood that also meant "include the capability to display, modify and delete customer record". Maybe because as a tester I've had more experience dealing directly with end users, I had understood that the customer wanted the full "CRUD" - Create, Read, Update and Delete - range of functions. It made a huge difference in the estimate for that story! The tester can help the customer state exactly what he wants.

Make sure tasks needed for testing are covered as well. We've been short on our estimates sometimes because we didn't allow enough time for testing and test support. For example, you might need a script to load test data. You can assume responsibility for this, but it needs to be accounted for in the tasks.

Be ready with your own input.While estimating tasks, the developers have to start thinking about the design. If you see tasks reflect a design that isn't test-friendly, speak up. For example, it's much easier to test an application where business logic is on the server side. If you see a task for writing Javascript with business logic in it, squawk.

Second Gear: Writing Acceptance Tests

The Planning Game used up the first day of our iteration. This is typical. Since acceptance tests are the requirements for an XP project, you need to complete them as soon as possible in the iteration. My goal is to have them approved by the customer and delivered to the developers within a day after the end of the Planning Game.

In XP, developers write unit tests before they write code, and those unit tests have to pass 100% at all times. In addition, developers integrate their code often, sometimes several times a day. Each time our developers build on their own machine or the integration box, all the unit tests run, and if one fails, the developer has to stop and fix it. The quality of the code delivered to me for testing in XP projects is heavenly compared to what I was used to in more traditional processes, where unit testing wasn't strictly enforced and integration was an afterthought.

In my first XP projects, I spent a lot of time studying the unit tests to see how thorough they were and what gaps I might need to fill with acceptance tests. We use jUnit for unit testing. Though I'm not a Java programmer, the unit tests are easy to understand, run and even add to. Now I'm confident that the unit tests are quite thorough. I always review the acceptance tests with the developers. Sometimes they already have a unit test which covers an acceptance test, so there's no need to duplicate the effort. Other times they suggest that a particular acceptance test could be done more effectively as a unit test (sometimes a unit test is the ONLY way to perform a particular test).

For acceptance tests, I don't try to exercise every path through the code. With the fast iterations of XP, you can't afford to test beyond the minimum needed to be confident that the business value has been delivered. I focus on end-to-end testing of the application from a user's point of view. Most of the defects that I do find come from tests designed to break - bizarre sequences of actions or stress testing. The unit tests usually do a good job of covering the "happy paths".

Ideally, there is one customer on the project full-time who can pair with me for writing the acceptance tests. Sadly, it is not an ideal world; sometimes the customer is only available part-time. Worse, sometimes there are multiple people from the customer's company who come and go, don't talk to each other, and have a totally different perception of the project.

In any case, you have to pin the customer down at least for a short meeting and have her tell you what acceptance tests she wants. How will the customer know when the features she has requested work? How will the developers know they've completed the customer's stories? I've had good luck with extracting tests to prove *intended* functionality or "happy paths" from customers. More difficult is defining tests which will help the team avoid potholes and blind curves: What happens if the end user tries a totally unexpected path through the online system? What are the ways someone might try to hack past the security? The customer must also specify criteria for load and performance testing. If a system is expected to handle large numbers of users or transactions, or perform at a

certain speed, the developers' estimates may need to be higher. You may even need a story for system stability or performance. I also require customers to provide test data. I can make stuff up, but it might not look like what they envision for their production system.

Make your acceptance tests granular enough to reflect how much business value has been successfully delivered. If you have a test case of two hundred steps and one step fails, you have to fail the whole test case even though most of it worked. There's no hard and fast rule. On our last project, we had eleven stories and forty-three acceptance tests. This number is misleading, though, as some tests were run with several different sets of test case data. See Sidebar 2 for some guidelines on writing effective acceptance tests. Figure 1 shows a sample acceptance test. [[[I was wondering here how many tests you might expect to have. Is there one test per story? Are all tests driven by stories? What generalizations can you make about how much is enough or what types are right?]]]

Acceptance Criteria

Unlike unit tests in XP, acceptance tests don't necessarily have to pass 100%. I ask the customer to define which test cases have to pass for the iteration to be a success. In XP, the iteration must end on time - the end date is not negotiable. Developers can produce high-quality code with XP, but while I find far fewer defects than I was used to in my pre-XP days, I still do find a few, and they can be serious. If they aren't all fixed by the end of the iteration, we have to put them on a story card, estimate them and let the customer decide whether to fix those or go on to new functionality. The customer has the right to 100% acceptance test success; it just comes at a cost. Agreeing on the acceptance criteria up front protects both the customer and developers from last-minute misunderstandings. Since this is XP, the customer can change her mind, but that may mean a new story for a subsequent iteration.

Here's an example. We have a customer who needs to get a functional web application up to show their venture capitalists. The functionality is very important but right now, the customer doesn't care that much about error handling. A story specifies a standard error screen to be displayed for all errors. We write acceptance tests to test for the error screen in case of any error, but the customer deems it non-critical. In the course of testing, we find that a rather unnatural path through the screens causes an error to be displayed in a user-unfriendly manner. All the other acceptance tests passed. The defect goes on a story card and gets estimated. The customer may still feel it is trivial, or may have decided when she saw that hideous screen that it needs to be fixed right away. OK, fine, she can choose the defect fix story for the very next iteration.

This sounds pretty simple, but in practice I've found it pretty hard. It takes time for the customer to build trust in the XP team. They're inclined to freak out about even trivial defects and don't want us to go on to the next iteration until they're all fixed. The less technical the customer, the more likely this is to happen. Be firm and turn those bugs into stories. You want a happy customer, but you can't give away work for free, and you can't let iterations run past their end date or your project will degenerate into chaos. As

you get through more iterations and the customer sees the business value being delivered, everyone will feel more confident.

You may need to pair with both the customer and a developer to fully understand all the stories and complete your definition of the acceptance tests. Often, when discussing the tests with the customer, I realize there has been a major disconnect between the customer's intetions and the developers' understanding of a story. In this case, I have to get the customer and developers back together to ensure we're all on the same road. The developers might need to re-estimate the stories, and the customer might even be forced to drop a story from the iteration. Better this than delivery of something the customer didn't want.

Acceptance tests are living, breathing entities. You may think of a more effective test for a feature, or the customer may decide they want something a little different. Just like the code produced by XP, acceptance test definitions and any automated test scripts you develop are subject to continual change and refactoring.

Third Gear: Performing and Automating Tests

It's day three of the iteration and we have a set of acceptance tests defined and approved by the customer, with test case data to run through them. If it's the first iteration of a project, you probably don't have any code to test yet, but then if it's the first iteration you probably need more time to finish defining the tests. You may be able to do some pre-automation work: develop templates, even develop some scripts if you have, for example, mockups of the user interface screens. By the second iteration, it's much easier to jump into automation as soon as your tests are defined.

Until you automate the tests, obviously you need to run them manually. I usually test the first iteration manually. Run the acceptance tests as often as you need to. Each time the developers have a new build, run all the tests that can run. I've had manual tests which were very tedious and time consuming. For example, on one, I had to add items to the database through the user interface, run queries from shell scripts to verify the database, run monitors which used the items to gather data, run more database queries to make sure data was persisting, and view the monitor logs to verify the actions being taken, comparing timestamps with the database records. This does make you wish for automation - more on that in a minute.

Even if I test manually, I need reports of test results, which I try to post daily in the development room to help the team steer the project. We have an in-house tool that lets us drive manual tests with test cases defined in XML, and record results so that we can automatically generate pretty graphs of the results. Just because tests are manual doesn't mean they can't be comprehensive, repeatable and timely, with published results.

While the XP books say to always automate all acceptance tests, I believe this is something you need to evaluate. Automation isn't free. If I'm the only tester, there are only so many hours in my day. How many can I devote to maintaining existing tests and developing new ones? Recently we did some work for a customer which was only a small subset of their eventual application. Even this subset was complex and had many components which had to be tested through separate interfaces. It wasn't worth the effort, which would have been considerable, to automate all the tests for this project. Any user interface tests would have to be changed anyway when the production interface was implemented. At the same time, there was no way to test such a complex system and keep pace with the developers without SOME automation. In addition, some load testing was needed.

I wasn't going to get where I needed to go in time by taking the back roads - I had to get on the freeway some of the time. When developers are in high gear, I often need to run the tests several times a day. I automated to the point where I could test fast enough and not spend too much time maintaining the automated tests. I turned the scripts over to the customer with documentation so they could keep using them until the user interface changed so drastically it wasn't worth updating them.

Test automation practices for XP are an entire article in themselves, if not a book, but see Sidebar 3.

Rough Road Ahead

The first iteration of a project is generally the roughest stretch of road for the XP car. You probably won't get code delivered to you in time to automate much. Concentrate on your roles as customer advocate: making sure the customer gets what he's paying for, and developer guardian: making sure the developers don't have to give stuff away for free.

In subsequent iterations, developers can organize their tasks so that pieces and parts of the new functionality can be delivered to you starting early in the iteration. With each iteration, you can grow and refactor your automated test scripts so that you can continue to keep up with the developers.

A caveat: the XP tester cannot succeed unless the developers are strictly following the XP practice of test before code, with 100% of unit tests always passing, and continual integration. If they don't, one tester cannot possibly keep up with the work of 4, 6, 8 or 10 developers. If the developers don't grasp the importance of these XP practices and your big boss can't or won't enforce them, start hiring more testers. You're gonna need them, or your project that was headed to Mexico is going to end up stranded in the Mojave Desert.

Are We There Yet?

Acceptance test results are the mile markers along the XP highway. Each time I run acceptance tests, I produce a color-coded graph showing the number of tests written, run, passed and failed. See Figure 2 for a sample report. Early in the iteration, you'll have new tests for functionality that hasn't been delivered yet, so those won't be run. A lot of tests will fail. As the days pass, you look for the green "pass" bar to get bigger and the red "fail" bar to fade away. If that isn't happening, it's a sign that your XP car may have taken an unpleasant detour, and the team needs to figure out how to get back on track. It may mean that the team has taken on too much for one iteration and needs to ask the customer to reduce the scope. It may simply mean that the developers misunderstood a customer requirement and only a few quick changes are needed. Your acceptance tests, with results posted for the whole team to see, provide the landmarks: keep watching for them.

The customer may not have time to sit with you each time you run tests, but as you near the end of an iteration, it's time to get the customer to sit down and run through all the acceptance tests (and load tests and other tests as applicable). Let the customer be satisfied that acceptance criteria were met. If they come up with new standards for the quality they want, those can become stories for subsequent iterations.

Eyes on the Road

To help the XP team steer, you need to do more than write, automate and run tests. Every day at the standup meeting, where each team member reports tasks completed the previous day, tasks to be done today, and any problems they're having, ask questions. Pay attention to clues that the developers have underestimated or run into a roadblock. Each XP team should have a tracker (we call this role Goon) who finds out how much time is actually spent on each task and gives the team feedback on its progress. (Don't let anyone assign you to this role, your job is hard enough already). Being in a more detached role, sometimes the tester can see Deadman's Curve before everyone else. "Story XYZ has taken a lot longer than we thought due to the database problems. Do y'all think we need to talk to the customer about dropping a story?" In my experience, developers are eternal optimists. Testers can help with reality checks. If you completed 90% of all stories, the customer has nothing to see. Completing 100% of fewer stories is better. You have to finish the interation on time, and you have to the right to deliver high-quality software.

I Can See the Ocean!

When I was a kid, the best part of a trip was finally getting in sight of the destination. In my previous, non-XP jobs, delivery day was usually a nightmare - a time of panic, pointless bickering and late hours.

What a revelation the day we ended our first "real" XP project. Nobody worked late the

day before. On delivery day of the final iteration, we calmly wrapped up a few tasks and handed the system over to the customer with almost as much documentation (written as the system was produced, so it was actually accurate) as a traditional project. We even videotaped members of the team explaining the various deliverables to the customer, including Q&A sessions, and gave the tape to the customer as an extra aid to communication. We had time for lunch, and we knocked off early to go celebrate with a round of beers.

As a bonus, we delivered far more to the customer than they had asked and paid for. We didn't do any last minute hacks - we delivered quality software and documentation. To me, this is the ultimate destination of Quality Assurance and testing: the Grand Canyon, Niagara Falls and the Golden Gate Bridge all rolled into one. Rather than just a sense of relief at having made it without driving off a cliff, we were exhilarated from the journey . We took a wrong turn or two along the way, sure, but the scenery was great and it was fun to be able to drive fast and keep up a steady pace.

I won't tell you XP is all hearts and flowers, or all Foosball and Oreos, or whatever metaphor you prefer. There are days where I've hit a roadblock and the developers are too busy to help me, or I found some ugly defect through a load test and the developers think I'm hallucinating, or the customer is being a pain, and I feel terribly lonely and hate my job. Sometimes it's hard being the only tester. Usually all that's needed is a little patience, maybe work on some other task for a little while until someone is free to come pair with me and attack the problem. Most days, like just about every other XPer I've talked to, I actually look forward to coming to work. Sometimes, they even let me win a Foosball match.

SIDEBAR: XP Practices

Sidebar: Keys to Effective Acceptance Tests

- *Make them granular enough to truly show the project's progress. If you have ten tests of fifty steps each, there is a high probability that at least one step will fail and you could show 100% failure even though 90% of the functionality works fine. Better to have fifty tests of ten steps each which will pinpoint more exactly on the map how far along the route your team has driven.*
- *Separate the test case data from the actions. Most people are comfortable with a spreadsheet format, which can easily be converted to other formats for input to automated tools.*
- *Identify areas of high business value. Describe basic user scenarios which can be used for automation.*
- *Include "nasty path" or what Hans Buwalda calls "soap opera tests" - actions and paths through the system that only an end user could imagine. Many serious defects are found this way.*

- *Include criteria for load, performance and scalability if applicable. Customer must decide what is appropriate for each iteration*
- *[[[How do you know if you have enough? (That's the perennial question - maybe you could work the Developer Protector and Customer Advocate roles into the answer. You're done when you do an adequate job of those things.) Maybe there's no short answer better than "that depends", in which case saying nothing is better.)]]]*

Sidebar: Tensegrent's XP Test Automation Principles:

Automated test scripts must:

- *Be modular and self-verifying*
- *Verify the minimum success criteria[[[need brief description of what these are]]]*
- *Contain no duplicate code*
- *Do the simplest thing that works*
- *Feature reusable modules [[[I know what you mean, I think, but I'm not sure the reader will. Can this be put differently, in a few more words?]]]*

Tensegrent's XP Automated Test Practices:
- *Continually refactor*
- *Remember, captured scripts are murdered by change*
- *Verify most critical functionality with a "smoke test".*
- *Add to automated tests as long as maintenance budget not exceeded [[[neat idea]]]*
- *Produce reports which are visual, color-coded[[[is this relevant to only automated tests, or are manual tests handled the same way?]]]*
- *If test tools are needed that can't be purchased, they become stories too and must be developed as part of the project. Customer pays for automation.*

Sidebar: XP Tester Practices

Work in the development room, alongside the developers. If you're in the car with them, you'll know every turn they take, whether or not they understand the map, and if they're asking directions.

Pair Test. Academic studies [[[Has there been more than one?]]] have shown the value of pair programming. Pair testing has the same advantages. Ask a developer to pair with you to develop an automated test script. In XP, nobody can turn down a request to pair! Pair with the customer to write and run tests. You'll be amazed at your productivity and at how much more fun you have.

Represent the customer. Especially if the customer doesn't participate 100% of the time, look out for her interests. Make sure the developers understand the customer's needs and that their estimates include everything the customer intended. If the developer tries to say something wasn't part of the story but you know it was, stand up for the

customer. It's your job to make sure the customer gets what he pays for.

Protect the developers. You're doing the best job you can for the customer, but nobody gets something for nothing. If you find a big bug that wasn't part of the story, report it, but tell the customer it has to be estimated and included in a future iteration. Yep, this is a big gray area, you have to use your own judgement as to what is fair to everyone.

Make sure the team has enough snacks. XP teams need fuel in the tank. You spend most of your time making their lives miserable ("I found another bug!") so it's good PR to put out a cookie jar full of Double-Stuff Oreos or, even better, bring in brownies occasionally. If food isn't your style, it might help to be good at Foosball.

Brian - for figures, I could provide sample acceptance test format or sample test reports. [[[We should think about including both. I don't know if there's room. Of the two, we would probably go with the latter, if only because they add some visual interest - charts! - to an article that's mostly text.]]] Cartoon pictures of people driving, looking at maps, having road rage might be cute. I'll send along a cartoon I found. [[[The production people will probably take the metaphor and run with it. One article per issue is illustrated with cartoons (not clipart; we actually pay someone more than I probably want to know for custom work).]]]

Figure 1:

Figure 2:

	A	B	C	D	E
1		**Ref #:**	Template for acceptance test: Timecard/QA/AcceptanceTest.sdc		
2		**Iteration:**			
3		**Functionality proven by this test case * is / is not * critical**	**What does this test?** *Example: Tests to make sure that when a time entry record is input, it is saved to the database and the report generated correctly.*		
4		**What to do:**	*Examples given in italics*		
5	**Step**	**Command/URL**	**Action**	**Input Data**	**Expected Output**
6	1	*Access www/timecard/addEntry in browser*	*Select a project, release, iteration, task, and enter duration, date and comments*	*Row 1, columns Project, Release, Iteration, Task, Duration, Date, Comments*	*Selections displaying on screen*
7	2	*Still on www/timecard/addEntry*	*Click the save button*		*Message that data was saved to database*
8	3	*Access www/timecard/generateReports in browser*	*Select a project, start date and end date*	*Row 1, columns Project, Start Date, End Date*	*Report generated – data matches row 1 columns Project, Release, Iteration, Duration, Cost and Total Cost*
9	4	*Repeat steps 1 – 3 with each row in the test case spreadsheet*			

Figure 2 (these gifs won't stay where I put them, I don't know why. Anyway I might send you better ones later, just don't have time now).

Tests Written	5	5	5	5	5
Tests Run	2	5	4	4	5
Tests Passed	1	5	4	3	5