eXtreme Rules of the Road

QUICK LOOK

- The tester's role in eXtreme Programming
- Tips for planning and performing acceptance tests
- Navigating challenges of high-speed, iterative design

How a tester can steer an eXtreme Programming project toward success

by Lisa Crispin

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 customer needs. I'm not alone; a 1995 Standish Group study showed that only 16.2% of software projects are completed on time and within budget. But why?

Reading Kent Beck's book, *eXtreme Programming Explained*, was my epiphany. It reminded me of my days as a baby programmer. Instead of writing reams of specs, in those days we prototyped everything and reworked projects in short iterations, with our customers sitting alongside. Although that worked pretty well, as the years progressed I found myself working in more traditional waterfall methodologies, with varying degrees of success. I missed the immediacy and close contact with customers I experienced in my first job—and as I heard more and more people talking about XP, I recognized a set of practices that perfectly fit my own style.

Driving the XP Car

Kent Beck, a noted author on the subject, 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 sole tester on a ten-person XP team, I wondered who was driving the car, and how I could help make sure it arrived safely at its destination. What was going to keep the team from getting lost or making a wrong turn? Who was going to make them *stop and ask directions?*

The answer? 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, and letting everyone know how the journey is progressing. As tester, you're part of the XP team, yet you function with a level of independent objectivity.

Through trial and error, I've discovered some major contributions that a tester can make to help 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.)

[Acceptance tests] are the road maps for the iteration, telling the team where it needs to go and what landmarks to watch for.

Let's take a test drive through a typical two-week iteration (or short release cycle) of an XP project to examine the roles of an XP tester. The project we'll be working on is the first release of a Web application, a project just now beginning its third iteration. This is the stage in which the developers will fully implement features chosen by the customer, and you'll be defining and executing acceptance tests. This process will provide details about how the features should work, and you'll use the test results to prove to the customer that the requested functionality is delivered.

First Gear: Planning

We're on day one of this two-week XP iteration, and it's time to hold what Beck calls a "planning game" meeting with the customer, the tester, and the development team. We held our first such meeting at the very start of this release cycle, before we even backed the car out of the driveway, and we repeat that process (albeit on a smaller scale) at the start of each iteration. In these meetings the customer defines the "stories," which are 3×5 cards detailing the features or sets of functionality to be developed. The developers estimate for the customer how much work (measured by a point system) each story will take, and how many total points can be completed in this two-week iteration. The customer chooses stories that add up to that amount of effort and no more.

Assumptions are the first obstacles as we set out on the road. They're as hard to avoid as tumbleweeds on eastern Colorado highways, and they can do a lot more damage.

Customers often assume that their intentions are obvious. Conversely, most developers are used to being forced to rely on their own assumptions about what the customer wants, since in traditional software development the customer isn't always around.

As the tester, one of your most important jobs is to steer everyone away from making assumptions. Your best strategy is to ask *lots* of questions, especially during the planning meetings. If the customer says, "I want a security model so that members of different groups have different capabilities," you should ask, "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?" The customer may be assuming you'll know he wants the system to handle a hundred concurrent users, while the developers may assume that in this iteration you're focusing only on the functionality and not robustness.

Speaking from experience, it's better to ask now than have an unpleasant surprise for someone later on. You might consider assembling a checklist to go through: validation, error handling, load, security, uptime. Make notes on the story cards, the whiteboard, paper, in whatever form you can refer to later when you're writing acceptance tests. You won't think of everything, but the team will have better directions for their journey.

Stop, Look Both Ways, Listen...

Once our development team feels they understand the stories well enough, and each story has been assigned "effort points," we break each story into tasks. Everyone chooses tasks for themselves and estimates the resources each task will require.

During this time, you may notice that the developers have missed something the customer wanted. I remember one scenario (one I've seen repeated many times) where the customer wrote a story that said "screen for creating record." The customer assumed that my team would understand that she also wanted to read, modify, and delete records on that same screen. I got this, but because it slipped by the developers, their task estimates were off. We then had to explain to the customer that the revised estimate was bigger than originally stated, giving her two options: living with only the create screen for now, or changing her mix of stories to stay within the maximum points for that iteration.

While development tasks are being listed, estimated, and assigned, make sure tasks needed for testing are included too. Task and story estimates should include time for both testing and test support. For example, you might need a script to load test data. Even if you assume responsibility for this yourself, you still need to add it to the task list. Thinking about these tasks will help lead the team to a test-friendly design—and serves as a reminder that testing is the responsibility of the entire team, not just the tester.

Second Gear: Writing Acceptance Tests

Day two, with miles to go. Iteration planning used up the first day of our iteration, which is typical. So far, we have some index cards with "stories" in which the customer has summarized the functionality he wants. We've discussed these stories and drawn pictures on the whiteboard to help us understand them, but we haven't written down many details about them. Here's where the acceptance tests come in.

These acceptance tests incorporate the discussions of the planning game, as well as subsequent conversations between the customer and the development team. In effect, they become the requirements document for an XP project. They are the road maps for the iteration, telling the team where it needs to go and what landmarks to watch for along the way. Since these tests are so critical to the success of the iteration, they must be written as early in the iteration as possible. Your 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, acceptance tests are ultimately the responsibility of the customer. In fact, rumor has it that on some XP projects the customer actually writes the acceptance tests. My experience, however, has been different. Customers, especially external customers, are busy; they usually have a full-time *real* job in addition to working on your project. They don't have time to sit down and type up tests. Most customers aren't experienced testers, and probably can't write effective acceptance tests by themselves.

Get together with the customer for a short meeting and describe what acceptance tests are needed. If the customer is willing and available, have him pair with you to produce the test definitions and write the tests. I've had good luck getting customers to describe tests that prove intended functionality using the system only as it was intended—and never doing anything weird like clicking the same button twice or doing things out of sequence. But it's more difficult to define tests that will help the team avoid those unforeseen potholes and blind curves. What happens, for example, 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? It's the unexpected actions or data that often reveal problems.

The customer must also specify criteria for load and performance testing, if this is important for the current iteration. You should have discussed this in the planning game, but you need specifics for 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 separate story for system stability or performance.

Have the customer provide the test data for acceptance tests, so that it more closely resembles the production data. You both may need to be creative to come up with data designed to test error handling or negative conditions. Load test data probably has to be generated, but sometimes the customer already has large files of data they can provide.

Acceptance Criteria

Acceptance tests are designed to tell us when we've successfully completed the iteration's functionality. In XP, acceptance tests—unlike unit tests—don't necessarily have to pass 100%. So how do we know when we're "done"? As with any set of software development practices, we can't expect totally bug-free code, so we need some criteria to know when we can release software.

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 deadline is not negotiable. Given that constraint, the customer can demand 100% success for all tests—but this may come at a cost of higher estimates for the stories. Including non-critical tests gives the customer more flexibility. If they pass, they give the customer extra confidence; if they are still failing at the end of the iteration, the customer can decide whether to expend resources in the next iteration to fix the defects.

Test Overview

Acceptance test name	Login
Ref #	1
Story	6
Iteration	1
Critical functionality	Yes
What does this test do?	Tests the login screen, validation of user login, and password
Category	User management
Prerequisite	Records in rows 2, 3, and 7 in TestCase Sheet 1 are in database

FIGURE 1 Login test case overview

The Nuts and Bolts of Test Writing

Once you understand the customer's requirements for acceptance tests, you can start drafting the tests themselves. Remember, your goal is to have a good set of test definitions written by the end of the second day of the iteration. The goal of acceptance tests is end-to-end testing of the system from the user's point of view, not 100% coverage of every path through the code. In XP, you should do the minimum acceptance testing needed to verify that the business value required by the customer has been delivered. Some people hear the word "minimum" and think "insufficient." But minimum doesn't mean shoddy or incomplete. In XP, testers need the courage to define the minimum acceptance criteria, so that you can keep up with the pace of development.

We're in the third iteration here. The tests you wrote for the first two iterations might need additions, modifications, or deletions based on the new stories. You'll run the tests for already-implemented functionality as regression tests. Any tests that passed before must now get successful results every time.

As you write the test cases, there are some things to keep in mind. Your tests should provide quick, accurate feedback on how the iteration and project are progressing. Avoid lengthy tests with a lot of steps; if one step out of two hundred fails, you have to fail the whole test. Write concise, granular tests, keeping the action steps and the test data separate.

Figures 1, 2, and 3 show a sample acceptance test—consisting of a test summary, action steps, and test data—for a login screen. The test consists of only two actions, but they're repeated with eight rows of test data, some that should result in a successful login and some that should get an error message. You and your development team can probably write a script

to take the test data from spreadsheets such as these and put it in a format your test tool can read, which will facilitate automation.

Another good practice is to pair with a developer to review the acceptance tests. Together, you may find tests that might be better performed as unit tests. If there's a story for something only on the back end (e.g., implementing a data structure) that doesn't have a user interface story in this iteration to go along with it, unit tests may be the only sensible alternative. By reviewing the tests, the developer may find a disconnect between the customer and the team on a feature that needs to be resolved. This is the time to make sure that everyone's going the same direction. Even if the customer has to drop a story because one was underestimated, this is better than delivering something the customer didn't want.

Third Gear: Performing and Automating Tests

It's the third day of Iteration 3. You've added acceptance tests for this iteration to the tests already defined for previous iterations. The customer is satisfied that the results of these tests will tell him that all the stories completed by the end of Iteration 3 meet his needs. Next, spend some time preparing automated test scripts so that you'll be ready when the developers have some code ready for acceptance testing. You may have some manual tests from previous iterations that you now have time to automate. You can also make any needed updates to existing automated scripts from the first two iterations. Perform all these tasks with another member of the development team, switching partners frequently.

The XP books say to always automate all acceptance tests. This is ideal, and with a lot of projects it's doable. Still, sometimes the cost of automating all your tests will be higher than you can manage. How do you decide which tests to automate? Ask yourself how much time you can devote each day to maintaining automated tests, and how much time you have to do all your other work. Start by automating a basic script that covers the functionality most valuable to the customer. If you find you only spend thirty minutes a day keeping this script up to date as the software changes, and you have two hours a day to devote to test script maintenance, then add tests to your automated suite. If load testing is a priority, spend your automation budget toward that first.

In the race to automate, you can fall into a trap of oversimplifying tests. If you have a complicated manual test and don't have time right now to automate it, don't sweat it. Just make sure the manual test is repeatable, that it can be performed in a timely manner, and that the results are well documented.

Halfway There

Time passes, work proceeds, and now we're entering Week 2 of Iteration 3. The developers have completed enough tasks and even an entire story or two so that you can install and test some of the code. You may have some automated scripts that you couldn't finish without the actual screens, so you can complete those now. You run your automated and manual tests for this iteration and report any issues you discover. You run tests from the previous two iterations. You may find that there were changes to the software that require more updates of the older tests. As you run tests, developers may want to pair with you to debug problems your tests found.

STEP	COMMAND/URL	ACTION	INPUT DATA	EXPECTED OUTPUT
1	localhost:8080/login.jsp	Enter login name, password, submit	Login and password from TestCase Sheet 2	See expected result in TestCase Sheet 2
2	localhost:8080/login.jsp	Repeat step 1 with all rows in TestCase Sheet 2	TestCase Sheet 2	See expected result in TestCase Sheet 2

FIGURE 2 Login test case actions (abbreviated)

1	LOGIN	PASSWORD	EXPECTED RESULT
2	Testy	tester	Login successful
3	jim-bob	11111	Login successful
4	NULL	NULL	Invalid login and/or password
5	empty	(spaces)	Invalid login and/or password
6	bad (leading space)	password	Invalid login and/or password
7	:";'<>,.	{}|[]\^+@!	Login successful
8	longloginname12345678901234567890 12345678901234567890	longpassword12345678901234567890 12345678901234567890	Invalid login and/or password

FIGURE 3 Login test case data

This pattern continues the next few days. Each day you have more completed tasks and stories to test. Each time the developers have a new build to test, run all the tests that can run. Post the results in the development room so the team can see how close they're getting to the destination.

Your job isn't limited to acceptance tests. You'll most likely need to perform installation, compatibility, recovery, security, load, performance, and stress tests as well. Start running load and stress tests as early as possible in the iteration—they turn up more issues than anything else.

As you successfully complete your tasks, celebrate! The energy boost will help you keep your momentum.

Being in a more detached role, sometimes the tester can see a neck-breaking hairpin curve in the road before everyone else.

Avoiding Potholes

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 four, six, eight, or ten 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.

You need to follow XP practices too. Refactor your tests and automated scripts continually. Make them as streamlined and efficient as you can. Ask the developers to pair with you to develop tests or help narrow down a problem. It can be hard to ask for help; sometimes you might feel you're always having to ask people to interrupt their own work to do you a favor. Grit your teeth and ask anyway. You'll be more productive, produce better tests, and the developers will learn more about testing.

Keep the customer in the loop. It may be disruptive to you and confusing for the customer to sit with you on the initial test runs. You'll be discovering problems with your test scripts and refactoring your tests. But when a set of tests runs smoothly, invite the customer to come pair with you. He may notice something you missed. He'll feel good about seeing the progress being made, even if you're discovering defects.

Are We There Yet?

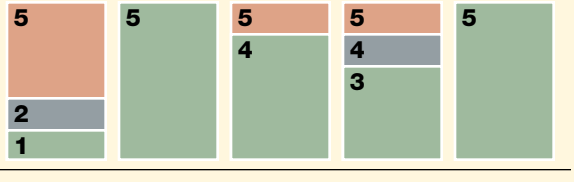
Acceptance test results are the mile markers along the XP highway. Each time I run acceptance tests, they produce a color-coded graph showing the number of tests written, run, passed, and failed (see Figure 4 for a sample report). Early in the iteration, you'll have new tests for functionality that hasn't 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 a 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.

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 "stand up meeting" (it's meant to be quick, and people don't dally as much when they have to stand up), each team member reports tasks completed the previous day, tasks to be done today, and any problems they're having. This is a good time to ask questions about potential roadblocks. Pay attention to clues that the work might exceed the original estimate for a task.

Each XP team should have a tracker who finds out how much time is actually spent on each task and gives the team feedback on its progress. That tracker helps make sure everyone's following correct XP processes.

Don't let anyone assign the tracker role to you; your job is hard enough already. But you can *help* the tracker. Being in a more detached role, sometimes the tester can see a neck-breaking hairpin curve in the road before everyone else. "Story XYZ has taken a lot longer than we thought due to the database problems. Do we 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. Face it: if you *almost* complete *every* story, the customer will see 100% failure. Completing 100% of fewer stories is



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

FIGURE 4 An acceptance test report

better. You have to finish the iteration on time, and you have the right to do good work.

Bugs on the Windshield

As described earlier, you and the customer agreed that bugs in noncritical acceptance tests are candidates for stories in the next iteration. While this sounds pretty simple, in practice I've found it difficult. It takes time for the customer to build trust in the XP team. He may be afraid to go on to the next iteration until all the defects, even trivial ones, are fixed. The less technical the customer, the more insecure he'll feel and the more likely this is to happen.

Another common problem is that issues come up that *look like* defects but are outside the scope of the iteration. It's one thing to imagine what a screen will look like and how it will work. When the customer actually *sees* the live software, he might wish that it worked differently from his original description, or might think of things he really wanted that he forgot to mention. This is where your acceptance criteria come in handy. If it wasn't part of the acceptance criteria, it doesn't affect the successful delivery of the iteration. Fortunately, it can easily be included in the next iteration. The customer won't have to wait more than two weeks at the most.

In our organization, we're still exploring alternative ways to handle outstanding defects to everyone's satisfaction. You want a happy customer, but there's only so much you can accomplish each iteration. If you get off schedule with the iterations, it will be hard to maintain your speed. Part of the solution to this problem is thorough up-front education of the customer, so that he has a good understanding of XP and how you deal with defects. As you get through more iterations and the customer sees the business value being delivered, everyone will feel more confident.

On the last two days of the iteration, have the customer sit down with you and run through all the acceptance tests (including load, installation, performance, and other tests you defined). 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. When you're done, the customer has new, "real live" software he can actually use, and you're ready to repeat the cycle for the next group of stories the customer chooses.

I Can See the Ocean!

When I was a kid, the best part of a trip was finally getting within sight of our 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 I ended my 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. We had time for lunch, and we knocked off early to go celebrate with a round of beers.

I won't tell you XP is all green lights and checkered flags. There are days where I've hit a roadblock in trying to get a test to work and the developers are too busy to help me, or I find some ugly defect through a load test and the developers think I'm hallucinating, or the customer is stressing about something, 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 working 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. STQE

Lisa Crispin (lisa.crispin@att.net) *is a Senior Quality Engineer at Agile Development LLC. While she managed to enjoy the 18+ years of her non-XP career by lucking into fun projects, she is thrilled now to embrace the challenges of being the tester on an XP team.*

STQE magazine is produced by STQE Publishing, a division of Software Quality Engineering.