XP Testing Without XP: Taking Advantage of Agile Testing Practices

Articles

Posted by:

Posted on : 2007/9/13 3:21:09

In the rocket-fast late 90s, I struggled with using traditional software process and testing practices on e-commerce applications. Then I read Kent Beck’s Extreme Programming Explained and had an amazing ‘aha’ moment. Ron Jeffries sums it up best: Extreme Programming is a discipline of software development based on values of simplicity, communication, feedback, and courage. It works by bringing the whole team together in the presence of simple practices, with enough feedback to enable the team to see where they are and to tune the practices to their unique situation. (www.xprogramming.com)

Applying more discipline to traditional waterfall process had not helped my team. A new kind of discipline based on values such as communication, simplicity and feedback might be the answer! I immediately took an opportunity to join an XP team as a tester, and enjoyed working on XP teams for the next 18 months. Indeed, XP did allow us to produce high-quality e-commerce applications on time and on budget! Our customers were happy! We were happy! We struggled some with how I, as tester, could best contribute to the team, as the XP literature at the time didn’t say much on that subject. With the whole team focused on quality and testing, we came up with some great solutions. I was so excited about XP testing that I co-authored a book on it (Testing Extreme Programming, with Tip House).

Author: Lisa Crispin, http://lisa.crispin.home.att.net

How XP Testing is Different

I found that XP testing was different in many ways from ‘traditional’ testing. The biggest difference is that on an XP project, the entire development team takes responsibility for quality. This means the whole team is responsible for all testing tasks, including acceptance test automation. When testers and programmers work together, the approaches to test automation can be pretty creative!

As Ron Jeffries says, XP isn’t about ‘roles’, it’s about a tight integration of skills and behaviors. Testing is an integrated activity on an XP team. The development team needs continual feedback, with the customer expressing their needs in terms of tests, and programmers expressing design and code in terms of tests. On an XP team, the tester will play both the customer and programmer ‘roles’. She’ll focus on acceptance testing and work to transfer her testing and quality assurance skills to the rest of the team.

XP Tester Activities

Here are some activities testers perform on XP teams.
* Negotiate quality with the customer (it’s not YOUR standard of quality, it’s what the customer desires and is willing to pay for!)

* Clarify stories, flush out hidden assumptions
* Enable accurate estimates for both programming and testing tasks
* Make sure the acceptance tests verify the quality specified by the customer
* Help the team automate tests
* Help the team produce testable code
* Form an integral part of the continuous feedback loop that keeps the team on track.

The Nature of XP Testing

The biggest difference between XP projects and most ‘traditional’ software development projects is the concept of test-driven development. With XP, every chunk of code is covered by unit tests, which must all pass all the time. The absence of unit-level and regression bugs means that testers actually get to focus on their job: making sure the code does what the customer wanted. The acceptance tests define the level of quality the customer has specified (and paid for!)

Testers who are new to XP should keep in mind the XP values: communication, simplicity, feedback and courage. Courage may be the most important. As a tester, the idea of writing, automating and executing tests in speedy two or three week iterations, without the benefit of traditional requirements documents, can be daunting.

Testers need courage to let the customers make mistakes and learn from them. They need courage to determine the minimum testing that will prove the successful completion of a story. They need courage to ask their teammates to pair for test automation. They need courage to remind the team that we are all responsible for quality and testing. To bolster this courage, testers on XP teams should remind themselves that an XP tester is never alone – your team is always there to help you!

XP Testing Without an XP Team

After my rewarding, if challenging, experience as a tester on XP teams, the bad economic times caught up with me and I had to take a job back on the "dark side". Oh, my new company was great and they were interested in XP, but they were a tiny shop with a huge job to do and it felt to me like barely controlled chaos. My initial attempts to apply XP practices, such as having daily XP-style stand-ups with my test team, were disasters. I just had to hunker down, hire people that I thought would be able and willing to try agile test practices, and wait for my opportunity.

Things eventually slowed down enough for my team to have time to spike test automation solutions. Many of our applications are written in Tcl. Tcl is a powerful scripting language which can lend itself to test automation, so we taught ourselves Tcl. We needed a way to load bulk data for volume testing, and one of the programmers helped out with a Tcl script; we took over maintenance for it and started expanding its functionality. Some of our test automation could be handled with fairly simple shell scripts. Other automation requirements demanded more robust tools. As we started down our own path of exploring test automation, the programmers started writing automated unit tests.

I learned the hard way that being an advocate for XP or any kind of agile development doesn’t introduce change. Instead, you and your team need to identify what hurts and look at what XP or agile approaches could help heal that pain. By adopting XP-style planning and writing acceptance

testing up front, my new development team has started exploring what agile practices might help them.

Let’s look at what agile testing practices might help you, no matter what style of software development process you are using.

Values

Earlier in this article I referred to the values by which XP teams develop software. These were defined by Kent Beck in "Extreme Programming Explained":
* Communication
* Simplicity
* Feedback
* Courage

Some people have added their own values to this list – the one I liked best is ‘enjoyment’. Extreme Programming is the only ‘official’ agile method I have actually practiced, but from what I have learned of agile software development in general, you can always turn to these values for guidance.

Communication is usually your first job. Whether you’re new to a position or you just want to try something new, you’re going to have to talk to your coworkers and your customers. Most importantly, you need to listen. As you proceed in your agile journey, remember the XP mantra: "Do the simplest thing that could possibly work". As a tester, feedback is your core function. You’ll need plenty of courage to try agile practices in a less-than-agile environment.

Let’s explore a typical project life cycle and see how XP practices can help you with testing, whether or not your development team is taking advantage of them.

Project Planning

Most projects begin with some kind of requirements gathering. The business experts may write business requirements. This is often followed by the development team producing functional specifications and/or technical specifications. If you’re working on a project which is producing documents like this, try to get in on the planning meetings. I know, the last thing you need in your life is more meetings. Requirements gathering meetings can be a great opportunity to put your agile skills to work. You’ll get a head start in understanding the project. By alternately looking at the requirements from the point of view of the end users, business experts and programmers, you can ask questions that flush out hidden assumptions.

Testing the Documentation

Whether or not you attend the meetings, you can add value to the project by ‘testing’ the documentation. This isn’t an XP concept, but it is linked to flushing out hidden assumptions during XP planning games. One way to test documents is to read through the document, writing a list of assertions. For example, maybe you’re reading requirements for an order entry system and have come up with these assertions:
* "The system will capture the customer service agent’s name".
* "The new system will be 50% faster than the old system".

* "The billing address will be required."
* "The zip code will automatically populate the city name".

Go over your assertions with the document’s author and clarify any ambiguous or contradictory assertions. Will the system capture the agent’s name by means of the agent logging into the system? What exactly in the new system will be 50% faster – the response time on each page? The time for placing an order? Do we have a baseline for the old system? If not, how do we quantify this requirement so we know it’s met? What about cases where multiple cities share the same zip code? Are there any optional fields in the address (such as the second address line)? Clearing up these issues now will save time for the programmers later, just as it does during an XP planning game.

If you’ve got a big project where the team sees a problem with finishing on time, see if there is a way you can suggest an XP approach to the planning. If the requirements can be grouped into story-like chunks, estimated by the programmers, and prioritized by the stakeholders, the team can take an iterative approach which will help ensure that some percentage of the necessary features are 100% ready by the deadline, even if less critical features are not yet implemented. If you’re new to agile processes or new to this team, it might not be the right time to take such a bold step, but keep it in mind.

Acceptance Tests

Write the acceptance tests (meaning any tests needed beyond unit and integration tests: system, functional, end-to-end, performance, security, whatever) during the planning phases of the project, rather than after development is complete. More importantly, if you’re trying an agile approach, write executable acceptance tests.

Executable acceptance tests serve two purposes. First of all, they’re a mechanism to involve your ‘customer’ or business expert in defining the tests which will verify that the level of quality and functionality he has specified is met. Secondly, they are a part of your automated acceptance test framework. If you can, avoid having to re-work your acceptance test cases into input to test automation tools.

There are quite a few test tool frameworks which allow this. The concept of data-driven testing isn’t new to XP, and there are plenty of tools which let you define tests in an Excel spreadsheet and then use these as input to your automated tests. XP-style test frameworks abound, as well. Here are a couple of sample JUnit tests:

assertTrue(login("bob","bobspassword"))

assertFalse(login("bob",""))

The first test proves that logging in with a valid userid and password works correctly. The second proves that logging in without a valid password fails. If you have fairly savvy business users, you can train them to read this syntax. If not, it’s not hard for your programmers to write a tool to suck data out of an Excel test case and input it to JUnit.

But wait a minute, you aren’t on an XP team where your programmers are totally on board to help you with acceptance testing. That might rule out Fit (http://fit.c2.com) and FitNesse (http://www.fitnesse.org) as well. Fit allows you to define acceptance tests as HTML tables, which are input to fixtures written in Java, Ruby or some other language that executes the tests. FitNesse goes one better by letting you define the Fit tests as a Wiki page, a sort of modifiable HTML page that’s even easier to handle than HTML. It will also let you define tests in an Excel spreadsheet and paste them onto the Wiki page.

If you really can’t get any programmer support to write the simple fixtures that drive these ‘behind the GUI’ tests, check out Canoo WebTest http://webtest.canoo.com/webtest/manual/WebTestHome.html., This allows you to define test cases in XML format. Again, users might not be totally comfortable with XML format, so you might have to do the spreadsheet-to-test tool conversion.

It doesn’t matter what tool you’re going to use. Defining acceptance test cases up front helps your programmers write the correct code. It gives you a huge jump on your testing too.

A word about test plans. Nobody ever reads them, so try not to write them. They’re cumbersome to write and impossible to maintain. Just write test cases. If your organization requires you to deliver a test plan, try to include information that will be useful to you, such as a release plan, and refer to the test cases as much as possible.

Technical Design

Depending on your technical skills, technical design meetings may be painful for you. Try to attend them anyway, if you have the opportunity. If you can even ask one question or make one comment that leads to a more testable design, you’re way ahead of the game.

XP has the wonderful benefit of making everyone on the development team conscious of testability. If you have to automate and perform acceptance tests, and it’s hard to do because your GUI is thick or hooked up to the back end in a way that you can’t test much behind the GUI, your next application is bound to be designed to be more testable. Unfortunately, your traditional development project won’t give you this golden opportunity. You can still raise these issues yourself (tactfully, of course) at design meetings. You can also continue to help facilitate communication between the business stakeholders and the programmers. You understand the application pretty well by now, and you probably are more familiar with the technical issues than the folks on the business side. Remember that communication value!

Executing Tests

Hopefully, you had the opportunity to write executable tests. If so, when code is delivered to you for testing, you’ve got a major jump on the test automation.

You’re not on an XP team, but maybe there are still ways you can benefit from the advantages of ‘pairing’. Collaborate with a fellow tester to spike some test automation solutions. Engage a programmer in brainstorming automation solutions.

The refactoring practice, changing code for the sake of maintainability or efficiency, is important for test automators. Take a little time every day to make little changes to your automated test scripts that will help you maintain them and run them effectively in the long term. You never really get the chance for that big modification, so take tiny steps as you go.

Start Early

Can the developers give you little bits and pieces to test? If you were able to convince them to take an XP-style approach to planning, this part is easy. They’ll develop testable chunks in short iterations and you can race along. If not, see what you can get delivered to you early. Maybe a programmer has a solution in mind but isn’t sure if it will scale well. Help by doing a performance test.

Involve the business experts as much as possible in testing. Keep simplicity in mind: What’s the simplest way to execute these tests? I hate to have to say it, but automation isn’t always the answer. You may have a one-time test, or you may need intelligent exploratory testing by a subject matter expert. Just remember to keep it as simple as possible! In most software projects, no matter what the methodology, you only have time to test the minimum.

Let Worry Be Your Guide

Risk assessments will help you and your stakeholders from worrying. Take a deep breath and determine what the minimum amount of testing will prove to the customer that the system behaves as specified. Lots of people have a negative view of the word ‘minimum’, but if it weren’t ‘good enough’, it wouldn’t be the minimum. Have courage: in most software projects, if you pull off the minimum testing, you’re golden.

XP builds risk assessment into the system with the planning game. The programmers always estimate all the stories and state their potential velocity, and the business experts or customers choose the ones most important to them that will fit into each iteration. With a more traditional project, look to a traditional risk assessment to help you. Identify risk areas, assessing the probability and impact of each. Work to mitigate the top one or two risks, then tackle the next one or two.

Retrospectives

Retrospectives are so important, some gurus such as Martin Fowler consider them to be one of the XP practices. Quality assurance is all about learning from history. Retrospectives let your team continually improve how you do your jobs.

Identifying Areas of Pain

There are different approaches to retrospectives (I much prefer this positive term to the oft-used ‘post-mortem’). Generally, your team looks at what went well, what didn’t go so well, what the team should start doing, what the team should keep doing, and what the team should stop doing. It’s sometimes hard to find time for retrospective meetings, but they’re a giant opportunity for the agile tester. I’ve found they are an ideal place to demonstrate how XP or other agile practices helped testing on the project. The goal of retrospectives is to identify

areas of pain and form an action plan to mitigate the top two or three. This is a great opportunity to look to agile practices for solutions.

In my case, over the course of many months, our test team built up credibility by doing a super job of manual testing. Then, we had a chance to show what value we could add with test automation. We did our own internal retrospectives, and demonstrated improvement by focusing on our problem areas. I also took every opportunity to expose the technical management and the programmers to agile techniques, by encouraging attendance at XP Denver meetings, distributing links and magazine articles, and taking advantage of an XP/agile consultant who offered a couple hours of his time to demonstrate test-driven development.

As a result, the leaders of the development team are curious about what agile practices might help them. Retrospectives are an ideal time to talk about these. Are requirements documents always inadequate, or take too long to produce? Writing acceptance tests up front, if you haven’t been able to do this up to now, is the ideal solution! This isn’t an article on how to introduce agile practices, but as a tester who benefits from XP testing practices, you can help your team improve quality by seeing what else it can borrow from the XP and agile world.

Celebrating Success

My favorite aspect of XP is how it celebrates success. When programmers or testers complete a task, they ring a bell or go play a round of Foosball (maybe nobody plays Foosball anymore in this economy!) I like to bring treats such as brownies to celebrate a release or just making it through a hard week. We’re often too busy going on to the next project(s) to notice, but it’s a nice lift for our team to go out to lunch or a little happy hour after a retrospective meeting to toast our accomplishments. Like all creatures, we do better with positive reinforcement than with negative punishment.

Perhaps the most important aspect of XP, which should be applied across all software development processes, is the one-team approach. Testers aren’t a separate, after-the-fact entity. They’re an integral part of the development team. As my buddy Mike Clark says, there is no "QA" in "team". Do whatever you can to help your whole team work towards delivering high-quality software on time.

Testing provides continuous feedback to help your whole team do their best work. Take baby steps if you have to – any progress is hugely rewarding! Above all, remember to follow Kent Beck’s advice, and embrace change! Originally published in the Summer 2003 issue of Methods & Tools