The Role of Testers in XP

Cem Kaner

August 13, 2003

These slides were originally presented as flipcharts at a workshop at Agile Development in Salt Lake, June 2003, then as slides at a meeting of Bay XP, Palo Alto, August 2003. I added annotations to reflect the oral presentations and discussion, and recently added a few additional thoughts. I thank Ron Jeffries, Chris Sepulveda, Ward Cunningham, and the participants of Agile Fusion for helping me clarify these ideas. As you will see, the thoughts are still tentative -- clarifying the puzzle rather than laying out the solution.

Research underlying these slides was partially supported by NSF Grant EIA-0113539 ITR/SY+PE: "Improving the Education of Software Testers." Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation (NSF).

Testing: The Traditional View

- Many years ago, the software development community formed a model for the software testing effort. As I interacted with it from 1980 onward, the model included several "best practices", such as these:
  - The purpose of testing is to find bugs.
  - The test group should work independently of the programming group.
  - There is a conflict of interest between the testing group (whose task is to get the software "right") and the programmers or project manager (whose task is to get the software to market).
  - The test group should have veto power over the release of a product to the customer.
  - Tests are designed without knowledge of the underlying code.
  - Automated tests are developed at the user interface level, by non-programmers.
  - Tests are designed early in development.
  - Tests are designed to be reused time and time again, as regression tests
  - To the maximum extent possible, tests should be automated. (I think this became a bigger deal in the 1990's than the 1980's).
  - Testers should design the build verification tests, even the ones to be run by programmers
  - Programmers *should* do extensive unit testing, but they probably won't, and so testers should assume that the programmers did a light job of testing and so should extensively cover the basics (such as boundary cases for every field)
  - The pool of tests should cover every line and branch in the program, or perhaps every basis path

Testing: The Traditional View (2)

- The old model included several "best practices", such as these:
  - Manual tests should be documented in great procedural detail so that they can be handed down to less experienced or less skilled testers, who will (a) repeat the tests consistently, in the way they were intended, (b) learn about test design from executing these tests, and (c) learn the program from testing it, using these tests.
  - A test design must include the expected result for the test. A test without an expected result is not a test.
  - There should be at least one thoroughly documented test for every requirement item or specification item
  - Test cases should be based on documented characteristics of the program, for example on the requirements documents or the specifications
  - Test cases should be idividually documented and, ideally, stored in a test case management system that describes the proconditions, procedural details, postconditions, and basis (such as trace to requirements) of each individual test case
  - Failures should be reported into a bug tracking system
  - A count of the number of defects missed, or a ratio of defects missed to defects found, is a good measure of the effectiveness of the test group.
- Each of these was sensible under some circumstances, but there's no magic in them.
- In 1983, I started writing *Testing Computer Software* in (negative) reaction to many of these, but I adopted and perpetuated several others.
- I think many of these practices will often retard quality-improvement. Others are inappropriate in the XP world.

Glass Box Testing in XP

- Unit-level change detectors.
  - These are unit tests, preferably written test-first, to help the programmer envision the code more clearly.
  - The objective of these tests is not to hunt bugs. It is to facilitate programming
    - Improve reliability, through clarifying detail-level requirements the implementation must meet
    - Support refactoring
    - Help subsequent code readers understand the intent and detail of the code
- Integration-level change detectors (e.g. using FIT)
  - Tests of communication between two (or more) routines.
  - Tests in which we set values of multiple variables or drive the program through multiple tasks or events
    - As with unit-level change detectors, the goal is not bug hunting, nor quality assessment. A tool like FIT might be used for those (bug-hunting / assessment, etc) tasks but that is a different use from change detection.
- **The practice**: the programmer creates change detectors to get insight into the problem and rapid feedback as she writes or modifies code. Run a test set (< 2 minutes) every compile, move additional tests into overflow and run them every few hours, or every day.

Glass Box Testing in XP

- Advantages
  - No roundtrip cost, compared to GUI automation and bug reporting.
    - The "roundtrip cost" is the processing cost of a bug, counting the labor of all involved persons. For example, the tester discovers a bug, rechecks her work, retests, simplifies, then files a bug report which is read and prioritized by a project manager, assigned to a programmer who debugs and fixes the bug, (or rejected/deferred and assigned for review by a triage team), returns it to the tester, who replicates and explores the bug fix before closing the bug. This can easily take 4 labor-hours and in more bureaucratic companies, might take 16 or more per bug. In contrast, to detect, debug, fix and retest a bug just after she has made the mistake, a programmer might only spend a few minutes. This is an enormous cost difference. If we eliminate roundtrip cost for hundreds of bugs, we can save the project thousands of labor-hours.
  - No (or brief) delay in feedback loop compared to external tester loop
    - Human factors research consistently concludes that short feedback loops are important to development of skills. The programmer who gets fast feedback about her errors is more likely to learn from them and connect her successful solutions back to the original problems she was trying to solve.
  - Support for experimenting with the language
    - (Added October 2003). I'm now teaching a test-driven development course for my second time. We use Eclipse as our development environment, doing TDD with Java and jUnit. I now have enough observation instances to be confident that I've seen a pattern. One of the problems in programming in a rich language (like Java) is that there are umpty-gazillion classes -- in effect, umpty-gazillion commands. No one knows them all. So part of the programmer's task is to experiment with new commands, to come to understand them while she's trying to figure out how to use them in production code. Inexperienced programmers have less knowledge, and therefore even more experimenting to do. As the tester becomes comfortable with TDD, she is likely to become more comfortable with creating throwaway test cases to try out new (to her) features of the language. This looks to be such a powerful learning-support structure that we're considering introducing TDD in our first year programming classes.
- Scope
  - Basic functional (and logic flow) testing, that looks for routine errors (see Marick's catalog).

Functional Acceptance Testing

- Story-based (aka use-case based)
  - In the main books on XP, including Crispin's book on testing in XP, story-based tests are described as the primary vehicle for functional acceptance testing.
- Typical tests are fairly simple
  - Happy path:
    - The main sequence(s) that lead to the result that the actor is trying to achieve
  - Sad paths
    - Error cases or other paths that don't achieve the desired result
  - Alternate paths
    - Non-error sequences that lead to results other than the typical, main result
- Might concatenate stories to develop more complex scenario tests
  - For extended discussion on scenario testing, see my paper on scenarios, to be posted soon at www.testingeducation.org/articles
- ***Comment:*** James Bach and I have spent an enormous amount of effort on the need to broaden the testing strategy of the typical test group. It's common for industrial test groups to use only one or two dominant test techniques. We recommend that they balance their efforts by using several techniques
  - see http://www.testingeducation.org/articles/blackbox_paradigms_tutorial.pdf

  because different tests are more effective for different types of bugs, or under different contexts.
  - see http://www.satisfice.com/tools/satisfice-tsm-4p.pdf
- Scenario testing is just one style (or main technique) for functional acceptance testing. I think that reliance this narrow is risky.

Functional Acceptance Testing

- One point of disagreement with XP literature
  - It's often said that *all* acceptance testing should be automated. I think this is a serious error.
    - There are significant cost/benefit tradeoffs associated with UI-level automated testing. The choice of all-versus-some automation should be pragmatic, based on the project's context.
  - I disagree that all non-automated testing should be fully scripted, so the person behaves as if s/he were an automaton.
    - This is another common recommendation (see, e.g. Crispin's book).
    - There are significant benefits to exploratory manual testing
    - The documentation cost of scripting is high and its effect on inertia (cost to future change) is high because of the cost of maintenance of the documentation
    - This treatment of a human is out of line with agile principles
    - It's an ineffective way to find bugs
      - In my courses on testing, I describe scripted manual testing as an industry worst practice. I came to this conclusion after working with scripted tests. I don't see why an agile development process would want to resurrect this paper-intensive process dinosaur.

Parafunctional Testing

- Functional testing is only part of the story. Consider these other attributes, which we call para-functional (or non-functional):
  - Security
  - Accessibility
  - Supportability
  - Localizability
  - Compatibility (configurations)
  - Interoperability
  - Installability and uninstallability
  - Usability
  - Performance
  - Scalability
  - And so on
- These are probably not well handled by customer stories.
  - These aren't well defined as a set of features. They are (or aren't) built into every feature. These also are often very technical--the customer is not likely to be an authority on scalability or interoperability in the way that she is on her own business processes.
  - One way to appear to handle this is with a broad "story" that says something like, "we want good security", but this is not a real story. You can't close it after a finite piece of work. And the customer can't provide definitive examples.

Eleven Dominating Black Box Techniques

This list reflects our (Bach / Kaner) observations in the field. It is not exhaustive. We put a technique on the list if we've seen credible testers *drive* their thinking about black box testing in the way we describe. A paradigm for one person might merely be a technique for another.

- Domain testing
- Risk-based testing
- Scenario testing
- Function testing
- Specification-based testing
- Regression testing
- Stress testing
- User testing
- State-model based testing
- High volume automated testing
- Exploratory testing

Let's Think About This By Examples

- Describe 3 projects and how they are being run
  - Comment:
    - We had an interesting discussion, but I don't think it summarizes well in a set of slides

The Location-of-Testing Puzzle

- For each of the quality attributes, who should do this type of testing?
  - No one
  - A programmer
  - A tester reporting to programmer-mgmt
  - A tester reporting to customer-mgmt
  - A customer
  - A consultant reporting to programmer-mgmt or customer-mgmt
    - Comment:
      - Discussions of testing in XP (Crispin's book is typical of this. I don't mean this to look like an attack on Lisa's book--I refer to her book because I think she accurately reflects the thinking of a lot of people in the XP community) focus on functional customer testing and pretty much ignore the parafunctional issues.
      - Ignoring them doesn't make them go away.
      - Saying they should be handled by specialists (as Marick does, http://www.testing.com/cgi-bin/blog/2003/09/25 - agile-testing-project-6) is a convenient rhetorical dodge but it fails to mesh with my experience. Most companies expect their test groups to handle most of these issues. They expect their testers to be generalists (see Marick's praise of "generalists over specialists" at http://www.testing.com/cgi-bin/blog/2003/06/30 - agile-context) with respect to the full range of testing issues.
      - I'm not ready to make a pronouncement as to who should do the parafunctional testing. I don't think it is a simple issue, I think the answer will vary with the local context, and that we'll learn a lot over the next few years about the ways this can work well in the XP contexts.
      - But I think it is foolhardy or disingenuous to ignore (or, especially, dismiss) these issues.

The Location-of-Testing Puzzle

HOW SHOULD WE DECIDE THIS?

- Factors to consider:
  - How technical is the work?
  - How much subject matter (customer application) knoweldge is involved compared to technical knowledge?
  - Who is in the best position to supervise the work?
  - Who is in the best position to mentor the person doing the work?
  - Who is likely to have the skill to do the work?
  - How will the person doing this work grow over time?
    - Comment:
      - *The list above are some of the factors that I think will be relevant in determining who is best equipped*
        - » *To do, and*
        - » *To manage*

        the different types of parafunctional testing.

Human Development in XP

- The programming process advocated in XP provides these to the programmer automatically.
  - Development and visibility of skill
  - Rapid feedback
  - Detailed review of the work product
  - Honesty
  - Peer support
  - Broadening experiences
- How will / should these be provided for the testing work?
  - Comment:
    - XP is masterfully tailored to provide personal and career development for the programmer. Within the process, the programmer will grow as a matter of course.
    - XP was not tailored to provide personal / career development to a person who works as a tester (as distinct from programmer). I'm not only thinking of the person who works full-time as a tester. To the extent that a programmer switches back and forth between test-driven programming and acceptance testing (functional or parafunctional), my claim is that XP is not tailored to provide growth to this programmer in his work on acceptance testing.
    - My question is simple. How should the process in your company be adapted so that it provides support for the development of the testing skills / knowledge of the person doing testing, as it does for the programming skills / knowledge of the person doing programming.

Role Alternatives

- The programmers are the testers.
- The customer organization supplies the testers.
- A third role gets introduced to XP, the testers.
- The programmers (or the customers) hire consultants (specialists) and dispose of them at end-of-project.
- Comment:
  - We can localize testing in the organization in any of these ways.
    - I am not advocating any one of them.
    - I am asking people to consciously consider the alternatives.
- Comment:
  - As I already noted, I think that the idea of significant reliance on specialist consultants for parafunctional testing is unrealistic and undesirable.
    - This has come up many times, in a longstanding debate in the field, sometimes called "The "Not My Job" theory of testing that pushes parafunctional testing away from testers and toward other specialists. I generally associate this side of the issue with process traditionalists, and just don't see it as a fit with the vision of agile development.

Role Alternatives

- I think the answer will vary across projects, but I think that we should take care to avoid re-making the following mistakes:
  - the purpose of testing is to find bugs
    - *The purpose of testing is to provide information, under one of several competing information-gathering missions*
      - Find defects
      - Maximize bug count
      - Block premature product releases
      - Help managers make ship / no-ship decisions
      - Minimize technical support costs
      - Assess conformance to specification
      - Conform to regulations
      - Minimize safety-related lawsuit risk
      - Find safe scenarios for use of the product
      - Assess quality
      - Verify correctness of the product
      - Assure quality
- Comment:
  - I discuss this somewhat in my paper, What Is a Good Test Case? at http://www.testingeducation.org/articles/what_is_a_good_test_case_star_2003_paper.pdf
  - Some people like to draw a dichotomy of missions, between finding bugs and helping managers make the release decision, as if those were the only two alternatives. They are not. Test groups operate under many different information missions. These missions, these types of information, are not dependent on the development methodology. Whatever is the development methodology, if the company needs a certain type of information, it needs a method for obtaining it. And if the company will regularly need that information, it needs people who are skilled at collecting the information and who have a process for collecting and validating it.

Role Alternatives

- I think the answer will vary across projects, but I think that we should take care to avoid re-making the following mistakes:
  - the test group works independently of the programming group
  - tests are designed without knowledge of the underlying code
    - *Think in terms of what knowledge the tester SHOULD have rather than what knowledge the tester should avoid*
  - automated tests are developed at the user interface level, by non-programmers
    - *These are inefficient and high-maintenance. Many will be replaced with glass-box tests or API level tests. Others can be avoided via exploratory testing*
  - tests are designed early in development
    - *We should design tests as we need them*
  - tests are designed to be reused time and time again, as regression tests
    - *Change detectors, yes.*
    - *GUI level regression tests? Trade costs and benefits. What is the inertial result?*
  - Black box testers should design the build verification tests, even the ones to be run by programmers
    - *Be cautious about replacing programmer regression with black box regression*

Role Alternatives

- I think the answer will vary across projects, but I think that we should take care to avoid re-making the following mistakes:
  - testers should assume that the programmers did a light job of testing and so should extensively cover the basics (such as boundary cases for every field)
    - *Obsoleted in the context of XP*
  - the pool of tests should cover every line and branch in the program, or perhaps every basis path
    - *Absurd in black box testing*
  - manual tests are documented in great procedural detail so that they can be handed down to less experienced or less skilled testers
    - *Enormous inertia*
  - there should be at least one thoroughly documented test for every requirement item or specification item
    - *Is the emphasis on existence of one test (why only one?) or on the documentation? Does this focus us on the right issues?*

Role Alternatives

- I think the answer will vary across projects, but I think that we should take care to avoid re-making the following mistakes:
  - test cases should be based on documented characteristics of the program, for example on the requirements documents or the specifications
    - *Hopefully, this is considered obsolete thinking in XP*
  - test cases should be documented independently, ideally stored in a test case management system that describes the proconditions, procedural details, postconditions, and basis (such as trace to requirements) of each individual test case
    - *Inertial expense is enormous. What are the advantages?*
  - failures should be reported into a bug tracking system
    - *This is often a good rule, but it is subordinate to the overall process. The purpose of the bug tracking process is to get the right bugs fixed. Other objectives are normally (not always) secondary.*
  - the test group can block release if product quality is too low
    - *Absurd*
  - a count of the number of defects missed, or a ratio of defects missed to defects found, is a good measure of the effectiveness of the test group.
    - *Depends on group mission. This is often a weak measure, distracting us from the real contribution, or interfering with it.*