Maybe Testers Shouldn't Be Involved Quite So Early[1]

An essay by Brian Marick
marick@testing.com
www.testing.com

I, like others, make a distinction between Quality Assurance and Testing. QA people observe the organization to see what's actually going on. Using that information, they find ways the organization might work better, help it change, and monitor what happens. Corporate management is ultimately responsible for Quality Assurance (since they are the people with the budget and influence to cause change), but they may delegate their influence to a team with special training.

While QA is primarily about process, testing—my specialty—is about product. Whatever else a tester might do, she certainly eventually exercises the product with the aim of discovering problems a user might encounter.

This essay is about that "whatever else" the tester does. Testers like me agitate to become involved at the very beginning of the development process ("upstream"). I think that's often a mistake. We often participate in the wrong way.

I'm talking about requirements analysis, the early part of the project devoted to understanding the customer's problem. This early understanding will usually be wrong, in part, but it's still worth having. A finished requirements analysis may contain statements like this:

> The user shall be able to store at least 300 email addresses, associating each with a tag of her choosing.

It may also contain user stories like this:

> Betty has sent email. She later realizes that she'll probably send essentially the same email to other recipients later. So she stores the already-sent email somewhere as a 'template.' When sending it again, she chooses the template, supplies the new email address (or addresses), perhaps changes the text slightly, and sends it off in the normal way. She uses the template many times.

After requirements analysis (or not long before its end), the designers build a picture of a solution in the form of a prototype, first-draft user documentation, or specification. I'll call this the "specification phase," although—quite often—the large-scale architecture of the product is designed at the same time.

---

[1] A variant of this article was originally published in *Software Testing and Quality Engineering* Magazine (www.stqemagazine.com).

What do we testers often end up doing during requirements analysis?

We design tests. The type of tests we can design from requirements like "the address book will hold at least 300 entries" are trivialities like this:

> Check that the address book can hold 300 entries, where all fields in each entry have the maximum number of characters.

We can get those at any time, and it makes no sense to keep track of the two obvious tests from that requirement until it's time to add the 120 other tests we'll get in the specification phase. Arguably, the designers might forget to consider that those 300 entries might be 300 *really big* entries—but is there any substantial harm in them not being reminded until the specification phase? The big risk here is that the designers will look at these pathetic few test cases and say, "*That's* the best testers can do? What dolts!" (I've seen it happen.)

We add requirements. Testers are users of the product. Our requirements are intended to allow us to extract product information that can't be gotten through the normal interface, or to control the system in ways that the normal interface makes difficult. But we'll have a better idea of what we need after we see what the normal interface does. It makes more sense for us to add our requirements during the specification phase. It will do no harm, since internal design will not have progressed very far.

We make estimates. As with requirements, we can do this better in the specification phase. Let's face it: estimates made even at the end of requirements analysis are usually off by a wide margin. (See Steve McConnell's *Rapid Development* for a summary.) Will a detailed analysis of requirements be of much help? We'll probably do just as well by asking how many developers there will be, discussing major new risks and project characteristics for a short time, then basing a rough estimate on past history. At the end of the specification phase, we can do a better job, perhaps using the process Cem Kaner outlines in "Negotiating Testing Resources: A Collaborative Approach (http://www.kaner.com/negotiate.htm)."

We do clerical checking. Rodger Drabick has an article titled "On-Track Requirements" in volume 1, number 3, of *Software Testing and Quality Engineering* magazine. It talks about his testing team's efforts to improve requirements by checking whether they're consistent, unambiguous, and verifiable by testing (among other things). I'm sorry, but these are clerical tasks. Designers should be trained to recognize that a requirement like "the transaction shall be processed within three days" is ambiguous (workdays or calendar days?). It's not that hard. You don't need to add extra people to do it. Oh, I concede that designers seem by nature less precise and meticulous than many testers, so they'll let more problems slip through. But are those the sort of requirements errors that—when discovered by testers in the specification phase—will make the designers' faces go white? I think not.

Enough examples. I'm saying that most projects can reasonably defer these add-on activities. Deferring saves money without hurting quality.

So have testers no skills useful in requirements analysis? Quite the contrary. Really good testers have a knack for speedily understanding the user's context—what tasks she performs and what misconceptions she's likely to have. Some of us will be better at constructing user stories than most designers are. And we work cheaper, too.

Testers can be useful as replacements for, not additions to, skilled designers. If untrained marketers create all your requirements (horrors!), you'll benefit from a tester on the team. If your designers are weak and a particular tester has a compensating strength, use her. If good designers are scarce, you can swap in a good tester.

But be careful. Does she have a track record of successful design in any field? If not, has she actively sought out indirect experience? For example, has she read Gause and Weinberg's *Exploring Requirements*?

Has she demonstrated an understanding of the central importance of the user's perception? Some testers use their knowledge of a user's actions only to discover clear-cut deviations from spec. I'd be much more comfortable using one with a track record of filing good bug reports on usability.

Does she have the trust and respect of the other designers? That's especially difficult for testers (unfair, but true), and it can't be decreed by management. When she filed those usability bug reports, did she convince anyone or were they rejected with the classic comment "works as designed"? Does she have the experience to envision at least some of the architectural implications of requirements? (Her fellow designers will likely discount her if she cannot.) Has she proven herself the kind of tester that developers ask for? (As James Bach says, "it may be that the best way to get involved *early* in a project is to have been proven valuable and trustworthy *late* on other projects.")

If your tester meets these criteria, give her a raise and welcome her upstream.

James Bach, Rodger Drabick, Dave Gelperin, and Jeff Payne made useful comments on a draft of this article. Alyn Wambeke was the copyeditor. Any grammatical or stylistic goofs were inserted in my post-publication edit.