Tester, Know Your Product

Fiona Charles

Originally published on TechWell.com and Stickyminds.com November 18, 2011.

What's your product as a tester?

It's an important question. Your product should determine your means of production—the essential activities and tasks, how you apportion effort, and the priorities you assign to various activities.

Your product is the principal thing that your stakeholders are paying for you to produce. For a tester, that product is information. It's what you produce that actually furthers achievement of your stakeholders' goals for the project or business. Yet, judging by effort, it seems many testers believe their product is a large set of documents that describe what they think they're going to do when they finally get to engage with a software system.

You could argue that you have more than one product—that information is primary, but you must still produce both evidence that your primary product is sound and guidance for your team or for people who might have to tackle this job at another time.

And indeed, that's true for many testers and test organizations. So I'll ask another question: Do you really believe that all the documents you currently produce represent the best use of your time in developing your primary and secondary products?

If you are in the testing mainstream doing what is sometimes called "structured testing" (by people who don't understand that most exploratory testing is structured) and if you are not working on a genuinely agile project, you are probably producing some or all of the following:

- A test project schedule
- A master test plan
- A test strategy
- Test plans (one for each major type of test)
- Test cases
- Scripts
- A test execution plan

As a consultant, I work with many organizations where all these documents are required deliverables of the standard test process before actual testing begins. With few exceptions, I find that testers and test managers expend excessive amounts of time on heavyweight documents that do little or nothing to move their testing forward, and that frequently even impede testing by diverting effort that would be better spent thinking about and working with the software.

Project schedules are important, but how much time have you spent messing about with a tool like Microsoft Project, vainly trying to produce a credible schedule that reflects your real intentions yet satisfies a project manager who demands an unmaintainable level of detail? Compare that to the time you've spent actually planning—thinking about what to do and how and when to do it. On some projects, the test manager no sooner produces a schedule than she has to turn around and rejig it because someone else has slipped his schedule or the project manager wants to "compress" some of the testing. I'm with US general (and subsequently president) Dwight D. Eisenhower, who said, "In preparing for battle I have always found that plans are useless, but planning is indispensable."

Testing is research. It's far more like battle than it is like manufacturing. We go into it with a strategy and plan in mind, but we don't know exactly what we're going to encounter that could completely derail our plans and force us to rethink. It's the thinking that's significant, not the probably-obsolete-tomorrow artifact that encapsulates it. Let's not spend more time churning out that artifact than it's worth.

Some form of schedule is likely essential, but can you say the same of a master test plan (MTP), particularly if you're also going to write a test strategy and one or more test plans? Often, an MTP is demanded far too early. How can you develop any plan before you understand the testing problem and your strategy for addressing it? The "solution" in most organizations that produce one is a boilerplate MTP copied and pasted from previous projects. Is copying and pasting useless text a cost-effective use of your time?

I've written in a previous article about massive test strategy documents devoid of strategic thinking and chock-full of copied text, either from other projects or from the MTP, which was copied from ... well, you get the idea.

Then we have supposedly more-detailed test plans, one for each type of test, often containing detail copied from the test strategy along with—we hope—some useful information about what you actually plan to do and what you need to accomplish it, now that you've had time to know something about it.

While the test manager is going prematurely gray over all these documents and schedules, the testers are off developing "test cases." Ideally, they're exercising creativity and superior analysis skills to generate test ideas, which they then capture as high-level test cases and approaches. But too often, they're locked into looking only at documented requirements and spending vast amounts of time diving into minute detail, scripting and typing expected results down to the point-and-click level. Have you noticed how much redundant detail test scripts contain? "Test steps," input data, and expected results get copied and pasted from test to test and script to script.

Why is this useful? It might be necessary if you're developing performance test scripts. But for manual testing, no skilled tester needs anything like this level of detail. And if you aren't working with skilled testers, why not?

Like plans, test cases and scripts are frequently outdated by the time you start testing, because the requirements or the GUI have changed.

All these standard, old-fashioned documents are heavyweight, inflexible, repetitive, and difficult to read. They are very expensive to produce, and their value is often

questionable. Should you really be spending more of your stakeholders' money preparing for testing than you spend working with the software and developing your product—the information they need you to produce?

I'm not saying that all test documentation is useless. Rather, I'm saying that the thinking tester or test manager should decide what is essential to the building of her real product in each given circumstance. Rather than manufacturing acres of continuous prose in boilerplate documents that almost no one reads anyway, think about the minimum set of artifacts you need to:

- Capture your significant thoughts and ideas
- Guide (yourself or others) in moving critical work forward
- Inform your stakeholders so they have opportunities to provide feedback
- Demonstrate due diligence for the record (where this is required)

Perhaps you could do all the plans in one thin document—say, a test strategy and high-level plan. Or, why not try using alternative, lighter-weight media like mindmaps and diagrams?

Instead of complaining that the written requirements aren't testable or sufficiently detailed or that they're changing too much, try spending your time thinking about and listing test ideas that will spark other ideas when you eventually get your hands on the software. Stakeholders may believe they need detailed scripts to review before you test, but it's actually easier for them to understand and review test ideas and test conditions. Many testers are now using mindmaps for test ideas and cases.

Throwing out the standard big documents in favor of optimal test artifacts won't be easy. Project managers, quality assurance people, and other people who don't understand testing are going to resist. So, why not start small? Talk to your stakeholders about what your product really is. Try a reduced documentation approach on one small project first, and then gradually extend it. Your stakeholders might really like the savings and the sharpened focus.

Let's take testing back! Knowing your product will show you the way.