

## Requirements and Testing: Seven Missing-Link Myths

Dorothy Graham

*Testing expert Dorothy Graham asserts that we can save a great deal of time and money if testers are involved in testing requirements. If the requirements have some consistent quality criteria, testers can raise questions and find problems before we turn them into code. —Suzanne Robertson*

**A** strong link between testing and requirements engineering can benefit both sides, but often this link is missing. Let's examine the seven most common myths or misconceptions behind this missing link.



### **Myth 1: Requirements at the beginning, testing at the end**

"We don't need to think about testing yet. Let's just concentrate on requirements." This attitude guarantees that you will have a rough time at the end of your project. Of course, getting good requirements is important, but getting testers involved during

requirements analysis is one of the best ways to ensure good requirements.

As Figure 1 shows, you should perform test design activities as soon as there is something to design tests against—usually during the requirements analysis for system and user acceptance tests.

A common problem, especially for testers, is the impact of late changes on requirements. Suppose you're in the last weeks of system testing, and user acceptance tests are scheduled to start running in two weeks. Suddenly, your users say, "By the way, we'd like the system to do this differently," which is extremely frustrating and happens more often than it should. But, what are the users really doing?

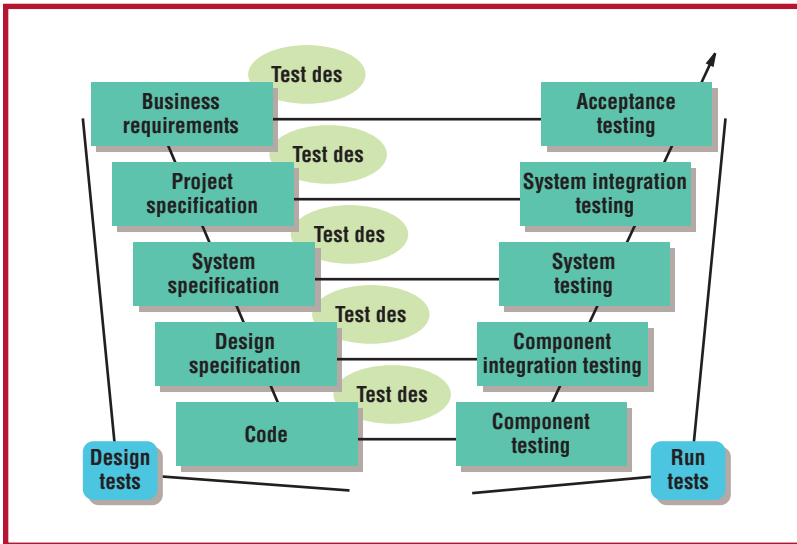
They are designing their acceptance tests. The act of test design highlights what they really want the system to do. If they had designed their tests early (left side of Figure 1), they could have discovered these problems before they were built into the system.

Getting users involved in both requirements and testing is crucial. Have you ever bought a used car? Would you go to the salesperson and say, "You know more about cars than I do, so take the test drive for me"? If you do, you deserve what you get. Similarly, users often say to techies, "You know more about computers than we do, so do the acceptance testing for us." *Caveat emptor!*

### **Myth 2: Testing isn't possible until the system exists**

"We can't do any testing because nothing has been built yet. Testers just play with the system and see what happens. Anyway, you can't test a piece of paper." Three things are wrong with this sentiment.

First, testing is more than just seeing what happens. It's far more rigorous and systematic than that. Second, it's more than just executing tests. As Figure 2 shows, executing tests and checking results is part of the fundamental test process, but other important activities exist as well. Third, you can and should test written requirements documents against business or project objectives for completeness and correctness. If you don't test requirements on paper, you



**Figure 1. A V-model for early test design.**

will build errors into the system that you could have easily removed much earlier.

**Myth 3: Requirements are used in testing, but not vice versa**

“You don’t test requirements—you test from them.” A tester’s mindset differs from a developer’s. It’s fairly easy to write a requirement that’s a bit vague and ambiguous but appears to be OK. However, when good testers look at a requirements specification, they devise specific test cases to flush out vague, ambiguous, or unclear requirements.

When you try to explain an abstract concept to someone, you say “for example” and illustrate the idea with concrete and specific cases to clarify the concept. Tests are the “for examples” for requirements. Think of “what if” use cases or business scenarios. If you consider how a particular user will use the system, the system functionality that seemed abstract when first described becomes specific for that particular user. Both testing and re-

quirements analysis benefit from having this feedback loop in place. Good requirements engineering produces better tests; good test analysis produces better requirements.

**Myth 4: If writing tests is difficult, it’s solely a test problem**

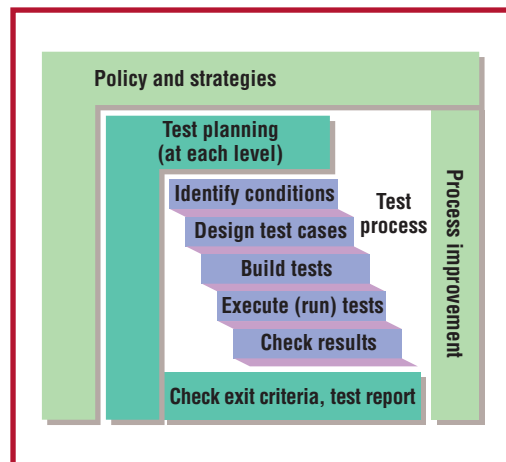
“The testers seem to have problems writing tests from our requirements—maybe we should get some better testers.” Not all requirements are created equal from a tester’s perspective. Specifying tests for some of them is easy; for others, identifying exactly what the system should do (and thus identifying tests to verify that it can do these things) is a nightmare.

Specifying testable nonfunctional requirements such as usability or performance is difficult.<sup>1</sup> Phrases such as *easy to use*, *user friendly*, *very reliable*, or *acceptable performance* are not specifications: they are vague, ethereal intentions. I subscribe to Tom Gilb’s law: “Anything can be made measurable in a way that is superior to not measuring it at all.”<sup>2</sup> Note that it is not made measurable in a perfect way but a useful way. Suzanne and James Robertson’s “fit criteria” show specifically how to make requirements measurable and therefore testable.<sup>3</sup>

**Myth 5: Minor changes in requirements won’t affect the project (much)**

“Just add a couple more spaces to this input field. There’s plenty of room on the screen. It’s a minor change; you won’t need to test it because it’s so small.” A change that appears to be minor from a requirements viewpoint could have a far-reaching impact, especially in testing. Suppose that adding two more characters to a field means that the database where this field is defined must be rearranged. What if this field corresponds to similar information held elsewhere in the system? What if the routines that check this field also check other fields that you have not increased in length? Do you now need two checking routines?

You must test all such changes to confirm that the system is now doing the right thing in every situation affected by this change. In addition, some unexpected side effects could arise, so you should also do regression testing to ensure that the system has not regressed in any other areas. How much testing you do depends on the risks of a change having both known and unknown impacts on the system. Testing can also help



**Figure 2. What is “testing”?**

mitigate the risk of change by giving confidence that such impacts are low.

**Myth 6: Testers don't really need requirements**

"I know we don't have good requirements for this system, but test it as best you can—just see what the system does." A tester's job is to adjudicate between what a system does and what it should do. The system should help the business accomplish a goal, so what the system actually does should be compared with those goals.

There is an *oracle assumption* in testing (which has nothing to do with databases or the companies that supply them—rather, it is based on the oracle of Delphi, who could predict the future with unerring accuracy). The oracle assumption states that the tester routinely knows what the correct answer is supposed to be, which is fundamental to testing. A test comprises the test inputs, preconditions, and expected outcomes. How do you know what the expected outcome should be? A requirement specification is this test basis, or oracle. So, yes, testers do need requirements; otherwise, you could argue that it's not really a test.

**Myth 7: Testers can't test without requirements**

"Because requirements form the basis for tests, obviously we can't do any testing until we have decent requirements." This is also a common tester's misconception. Sometimes changes are made to systems where requirements are inadequate or nonexistent. This makes the testing more difficult, but you shouldn't just throw your hands up and say that you can't do it.

Without requirements, testers still need some kind of test oracle—maybe users who are familiar with the way the system works, the old system, user manuals, or the tester's own opinions. What happens if the test oracle is the tester? Instead of merely comparing a system against a document, the testers are judging the system against their personal view of what it should do. (Actually, testers should always do some of this anyway, but that's another story.)

Some would say that without a specification, you are not testing but merely exploring the system. This is formalized in the approach known as *exploratory testing*, designed for situations with inadequate requirements and severe time pressure.<sup>4</sup>

If you end up with a good set of testware,

test plans, test specifications, and so on, the testware actually becomes the only requirement documentation in the project. So with good enough tests, who needs requirements at all? Do the testers become the requirements engineers? Is this a good idea?

If requirements are poor or nonexistent, testers must do the best testing they can under less than ideal circumstances. Testing is far more rigorous if it is based on good requirements, and tester involvement early on can help produce them.

hope I've convinced you that testers have much to offer in producing better requirements. Here's how to achieve it in practice:

- Invite testers to participate in requirement reviews and inspections
- Begin planning testing in parallel with requirements analysis
- Ask for sample test conditions and test cases to use as examples in the requirements specification
- Include in the requirements document any specific cases that you had in mind when analyzing requirements
- Use business scenarios and use cases to give specific examples of how the system should work
- Set measurable criteria for both functional and nonfunctional requirements

Testing is challenging in two ways: it's a rewarding intellectual activity, but it also challenges whatever the tests are based on. Make the link between requirements and testing. If you accept and encourage the challenges that testers make to your requirements, you will avoid these misconceptions and will end up with significantly better requirements and tests. ☺

**Acknowledgments**

Thanks for comments on drafts of this article from Clive Bates, Mark Fewster, Robin Goldsmith, and Dot Tudor.

**References**

1. B. Lawrence, K. Weigers, and C. Ebert, "The Top Ten Risks of Requirements Engineering," *IEEE Software*, vol. 18, no. 6, Nov./Dec. 2001, pp. 62–63.
2. T. Gilb, *Principles of Software Engineering Management*, Addison Wesley, Boston, 1988.
3. S. Robertson and J. Robertson, *Mastering the Requirements Process*, Addison Wesley, Boston, 1999.
4. C. Kaner, J. Bach, and B. Pettichord, *Lessons Learned in Software Testing*, John Wiley & Sons, New York, 2002.

**Dorothy Graham** founded Grove Consultants in the UK, a company that provides consultancy, training, and inspiration in all aspects of software testing. Contact her at [dorothy@grove.co.uk](mailto:dorothy@grove.co.uk).

**Good requirements engineering produces better tests; good test analysis produces better requirements.**