Risk and Requirements-Based Testing

James Bach, Independent Consultant

In *Reframing Requirements Analysis* (*Computer*, Feb. 1999, pp. 120-122), I proposed that requirements development is not something that happens all at once at the start of a project. In real life, requirements are negotiated in the course of two simultaneous dialogues throughout the project life cycle. Those dialogues entail asking, "What do we want?" and "What can we build?"

The quality of these dialogues goes a long way to determining the ultimate quality of the product. After reading that column, Jerry Weinberg e-mailed me the gentle criticism that the example I used (a file conversion tool) was too simplistic to illustrate the requirements testing problem effectively. What about safety-critical or otherwise complex products? Since Jerry co-authored, with Don Gause, my favorite book about requirements, *Exploring Requirements: Quality Before Design* (Dorset House, 1989), I feel compelled to address that problem and expand upon the role of testing in requirements development.

I define requirements as the set of ideas that collectively define quality for a particular product. I define testing as a process of developing an assessment of product quality. First, I'll reframe the requirements testing problem in terms of risk, and then I'll show what happens in complex or high-risk situations.

Editor: James Bach, 1198 South Fork Dr., Front Royal, VA 22630; voice (540) 631-0600; fax (540) 631-9264; j.bach@computer.org

I think we need to break out of the mythology that testing is some kind of robotic process.

RISK AND REQUIREMENTS TESTING

There are at least four alleged truisms about testing a product against requirements. Most of the testing textbooks on my bookshelf promote these principles, and each principle reflects some truth about the dynamics of testing.

1. Without stated requirements, no testing is possible.
2. A software product must satisfy its stated requirements.
3. All test cases should be traceable to one or more stated requirements, and vice versa.
4. Requirements must be stated in testable terms.

When we think in terms of risk, however, I believe a richer set of ideas emerges.

Testing without stated requirements

If it is very important to satisfy a requirement, and it is the job of the tester to evaluate the product against that requirement, then clearly the tester must be informed of that requirement. So there are situations where this statement is basically true.

The deeper truth is that stated requirements are not the only requirements. Because of incompleteness and ambiguity, testing should not be considered merely as an evaluative process. It is also a process of exploring the meaning and implications of requirements. Thus, testing is not only possible without stated requirements, it's especially useful when they're not stated. I think we need to break out of the mythology that testing is some kind of robotic process. Tremendous value comes from testers and developers collaborating. Skilled testers evaluate the product against their understanding of unstated requirements and use their observations to challenge or question the project team's shared understanding of quality.

A good tester stays alert for unintentional gaps in the stated requirements, and works to resolve them to the degree justified by the risks of the situation.

Satisfying stated requirements

The idea that a software product must satisfy its stated requirements is true if we define product quality as the extent to which we can reasonably claim that each stated requirement is a true statement about the product. But that depends on having a very clear and complete set of requirements. Otherwise, you're locked in to a pretty thin idea of quality.

The deeper truth is that while quality is defined by requirements, it is not defined as the mere sum of "satisfied" stated requirements. There are many ways to satisfy or violate requirements. Requirements are not all equal in their importance, and often they are even in conflict with each other. It unnecessarily limits us to think about requirements as disconnected ideas, subject to a Boolean evaluation of true or false.

A broader way to think about satisfying requirements is to turn our thinking around and consider the risk associated with violating them. Good testers strive to answer the question, "What important problems are there in this product?"

Tracing test cases to requirements

To the extent that requirements matter, there should be an association between testing and requirements. But talk about traceability so often boils down to a form of clerical Bingo: "For each requirements ID, list the test case IDs that relate; for each test ID, list the requirement IDs that relate." The completeness of testing is then presumably evaluated by noting that at least one test is associated with each requirement. This is a pretty idea, yet I've seen projects where this checkbox traceability was achieved by defining a set of test cases consisting of the text of each requirement preceded by the word "verify."

If the intent of the traceability principle is to demonstrate that the test strategy has validated the product against requirements, then we have to go deeper than checkbox tracing. We should be ready for our clients to ask the question, "How do you know?" We should be able to *explain* the relationship between our tests and the requirements. The fact that a requirement is merely associated with a test is not interesting in and of itself. The important thing is *how* it is associated, and that importance grows in pace with product risk.

Stating requirements in testable terms

It's important that requirements be meaningful. However, "testable" in this context is usually defined as something like "conducive to a totally reliable, noncontroversial, and observer-independent measurement that results in a true-or-false determination of compliance." Sometimes this point is emphasized with a comment that unless we are able to measure success, we will never know that we've achieved it.

To penetrate to the deeper truth, first recognize that testers, far from being drones, are blessed with normal human capabilities of discernment and inductive reasoning. A typical tester is capable of exploring the meaning and potential implications of requirements without necessarily being fed this information from an eyedropper like some endangered baby condor. In fact, attempts to save testers the trouble of interpreting requirements by simplifying requirement statements to a testable scale may make matters worse.

Here's a real-life example: "The screen control should respond to user input within 300 milliseconds." I once saw a test designer fret and ponder over this requirement. She thought she would need to purchase a special tool to measure the performance of the product down to the millisecond level. She worried about how transient processes in Windows could introduce spurious variation into her measurements. Then she realized something: With a little preparation, an unaided human can measure time on that scale to a resolution of plus or minus 50 milliseconds. Maybe that would be accurate enough. It further occurred to her that perhaps this requirement was specified in milliseconds not to make it more meaningful, but to make it more objectively measurable. When she asked the designer, it turned out that the real requirement was that the response time "not be as annoyingly slow as it is in the current version of this product."

Thus we see that the pragmatics of testing are not necessarily served by unambiguous specification, though testing is always served by meaningful communication.

REQUIREMENTS, TESTING, AND CHALLENGING SOFTWARE

I reformulate the principles above into the following, less quotable but more robust, guidelines:

1. Our ability to recognize problems in a product is limited and biased by our understanding of what problems there could be. A requirements document is one potential source of information about problems. There are others.
2. We incur risk to the extent that we deliver a product that has important problems in it. The true mission of testing is to bring that risk to light, not merely to demonstrate conformance to stated requirements.
3. Especially in high-risk situations, the test process will be more persuasive if we can articulate and justify how test strategy relates to the definition of quality. This goes beyond having at least one test for each stated requirement.
4. The test process will be more effective if requirements are specified in terms that communicate the essence of what is desired, along with an idea of risks, benefits, and relative importance of each requirement. Objective measurability may be necessary, in some cases, but is never enough to foster robust testing.

What happens when these principles are applied to high-risk or complex software? First, let me disclaim any and all concern about requirements and testing processes performed for reasons other than creating a quality product. For the purpose of this column, all the requirements documentation done simply to pass a process audit is of no consequence. Don't confuse that with what must be done to produce a quality product.

As risks and complexities increase, participation by testing in the requirements dialogue becomes more important if the test process is going to achieve its mission. More testing skill is needed, as is a better rapport with the development and user communities. In the dialogue about what we want, testers should seek multichannel communication: multiple written sources, diagrams, demos, chalk talks, and use cases. In the dialogue about what can be built, testers should be familiar with the technologies being used, and work with development to build testability enhancing facilities into the product.

Throughout the process, the tester should raise an alarm if the risks and complexities of a project exceed his or her capability to test.

There is nothing in the reformulated guidelines that suggests requirements must be made absolutely clear and precise. What these guidelines emphasize is the importance of managing the relationship between risk and a shared understanding of what quality means for your product. If you are on a challenging project and are managing risks and requirements, clarity and precision will emerge naturally. ❖