Quality Software

I suspect that, in the near future, many types of software will become commoditized, just as many types of computer hardware have. The open-source phenomenon is leading the way, with Linux and Apache ascendant on the Internet. Regardless of the motives of the partisans of open-source software, the motives of the important business users of these open-source applications are clear: They want cheap software with the same quality levels as the commercial alternatives.

Basic economics tells us, for commodities, prices and profit margins are low, features are standardized, and quality is an absolute must for participation in the market. Failure to deliver consistent quality damages a business' ability to compete in a commoditized market. To deliver quality software, we need to start with a working definition of what quality software is.

In addition to outsourcing, assessments, and consulting, my company, RBCS, offers training courses for software and systems professionals. Most of those courses focus on testing. We have taught thousands of attendees in dozens of countries around the world. Towards the beginning of these courses, we often ask people, "For the systems you build, what comes to mind when you think about the word, 'quality'?"

We usually separate the responses into two main groups: *outcomes* and *characteristics*. By outcomes, I mean what would the result be, after the software was delivered. By characteristics, I mean what would be true about the software that was delivered. Let's look at each group.

In the outcomes group, most attendee responses usually boil down to one of two definitions:

- The software conforms its specification.
- The software fits its various uses and purposes.

The first definition closely follows Phil Crosby's definition of quality, as given in his book *Quality is Free*. The second closely follows J.M. Juran's definition, found in his book *Planning for Quality*.

Juran's definition is my favorite. Fully articulated, it means the software has those attributes, characteristics, and behaviors that satisfy the customers, users, and other stakeholders, and has few if any of those attributes, characteristics, and behaviors that dissatisfy them.

The first definition sounds good initially, but turns out to be a will-o'-the-wisp when applied to software. According to Capers Jones' studies, almost half of all defects are introduced during requirements and design specification. Testing the quality of software against the specification only is like measuring with a flawed yardstick.

However, how do we measure against the "fit for use and purpose" definition, either? This is where the "characteristics" part of the discussion comes in.

Depending on the software or system in question, some course attendees list characteristics like reliability and performance. Some list usability and scalability. Some list data integrity. Interestingly, many fail to mention functionality; i.e., the ability to fulfill correctly the stakeholders' business needs for the software. When we mention that to attendees, the reaction is usually, "Of course!" It seems some people think that some quality characteristics—and, of course, the need to test them—are simply obvious.

Unfortunately, what's obvious to some people is not obvious to all, and what perhaps should be obvious to project participants is sometimes forgotten entirely. So, determining which quality characteristics are important, and how important they are relative to each other, is crucial to the proper focus of the testing effort. At RBCS, when we manage testing projects, we typically use a primarily risk-based testing strategy. In the RBCS approach to risk-based testing, we start by analyzing, for each possible quality characteristic, the various risks to the quality of the system. For each of these quality risks, we then determine what the level of risk is. This allows us to focus our test effort, and prioritize our tests, based on the risk posed to the system.

Of course, determining which quality characteristics are important, and how important they are, is not only crucial to testing, but also to the rest of the project team. Quality cannot be tested into software at the end of the project. Simply grinding out as many bugs as possible, in addition to being inefficient, will not result in software that yields the delightful quality that we experience with the most well designed products that we use.

So, where can you find a generic list of quality characteristics? Some companies use the ISO 9126 standard. This standard specifies six main quality characteristics—functionality, reliability, usability, efficiency, maintainability, and portability—and, for each characteristic, two or more subcharacteristics. For example, response time (performance) and resource usage are both subcharacteristics of efficiency.

I have found that a generic checklist of about two-dozen quality risk areas has worked well, too. At RBCS, we use this list to structure my conversations with project stakeholders about quality, particularly during quality risk analysis. What could go wrong in each quality risk area? How likely is that particular quality risk? How much trouble would it cause? Whether you use the RBCS checklist or the ISO 9126 standard, either will provide a framework for understanding system quality and how to test it.

This brings us to my final point. In about one presentation out of ten, someone will respond to the question about quality in a totally different way, giving a response that I would classify in a *knowledge* group. By knowledge, I mean how would you know whether the software had quality. A typical response in this group might be, "Software that was thoroughly tested in a way that covered all important quality risks, with few if any blocked tests, critical failures, or high-priority bugs at the conclusion of testing." These attendees understand that, while testing cannot change the quality of software, testing can offer the organization the opportunity to correct quality problems, and can build confidence where the system is observed to work properly. As a test professional who believes that testing plays an essential role in delivering quality products, I find this to be not only a good response, but a professionally gratifying one, too.

Author Bio

Rex Black is President of RBCS (www.rexblackconsulting.com), a leader in the area of testing and quality assurance. RBCS has over a hundred clients in about twenty countries around the world, offering them services like training, assessment, consulting, staff augmentation, and outsourcing. Rex's four books, *Managing the Testing Process*, *Critical Testing Processes*, *Foundations of Software Testing*, and *Pragmatic Software Testing*, have reached over 30,000 readers on six continents, but the penguins in Antarctica still won't buy them.

Resources

You can read more about quality risk analysis and risk-based testing in my books, *Managing the Testing Process, 2e*, *Critical Testing Processes*, *Foundations of Software Testing*, and *Pragmatic Software Testing*. You can also read the articles "Investing in Testing: The Risks to System Quality" and "Quality Risk Analysis" posted on the Library page of our Web site, www.rexblackconsulting.com. You can find the generic checklist of quality risks I mentioned on the Library page as well.