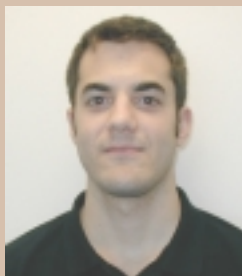


Putting Software Terminology to the Test

John Steven, *Cigital*

Project management relies on testing personnel's expertise to ensure software quality. However, contractual and management issues also determine a project's quality. Such issues might even control testing itself.

Consider the situation in which a client hires a vendor to build a piece of software. The client could specify the type, schedule, and extent of testing in its contract. Such terms might include compressing the time allotted to quality assurance or substituting certain tests for others. To successfully meet software quality goals, all parties involved must fully understand what testing the contract requires. However, testing literature, practitioners, and project management often have different understandings of common testing concepts. In this column, I describe two specific inconsistencies in testing vocabulary that can put a software project at risk.



Acceptance testing

A product vendor has a fiduciary responsibility to adequately test its software before release. Even so, a customer might also tie payment and liability-transfer to *acceptance testing* results as part of a vendor-customer contract. The contractual tie-in gives the customer leverage by requiring empirical evidence of quality before accepting and paying for the vendor's product, and AT becomes a decision point for the customer. Using AT this way helps the customer manage risk vested in the software development.

Unfortunately, customers who contract testing professionals for AT are often wasting time and money. The contracted test teams frequently lack the in-depth product knowledge required to provide much additional value over the system- or integration-level testing that the vendor used to harden the product. Furthermore, they frequently miss one of AT's key precepts.

In *Testing Object-Oriented Systems*, Robert Binder defines AT as testing on a system scope that a customer conducts to determine whether or not to accept a system. This initial definition only implicitly indicates—if at all—*what* should be tested. That, in part, is why test teams often end up repeating system testing and calling it AT.

Perhaps more specifically, William Perry, in *Effective Methods for Software Testing*, emphasizes that AT must be "...designed to determine whether the software is 'fit' for the user to use." He says that AT should not focus on requirements conformance alone. Rather, AT must draw on knowledge of the software's application to the business domain to determine its fitness to a particular set of users. (Inexperience in the business domain hinders AT more than any other type of testing and can render it ineffective.)

So, AT should focus on scenarios that users will likely encounter. Customers should consider AT as *positive testing*, or testing *happy path* (normal and thought-to-be working) use, rather than as more rigorous robustness testing. After all, the viability of features that a user won't encounter should have little impact on a product's acceptability.

Operational testing

The term *operational testing* can also be misleading. When project personnel hear the word “operations” they think of the individuals who run a deployed application on a day-to-day basis. Perhaps reasonably, OT has come to mean testing usage scenarios involving operations functionality in an application on a system level. Testing operations functionality is important to an application’s success and should be done in each round of testing. This is particularly true in financial arenas where humans undertake most of the work to complete and clear transactions through an operations interface.

However, the testing literature defined OT—at least as early as the 1970s—as late-lifecycle testing on a system level performed by users under normal circumstances. Another name for this is *beta testing*. For this, you gather usage from actual users by observing how they exercise the software under normal circumstances. A statistically valid beta test will pro-

duce more usage than you can manually analyze. Thus, the observed usages are filtered to determine which features of the software under test are used most frequently, and how.

The idea behind OT, as defined by the literature, is to submit an already well-tested software product to a controlled user base for trial and further testing. The goal is to focus quality assurance efforts on the software’s areas and features users most frequently encounter. This approach has obvious appeal: the project staff doesn’t need to guess where to focus test efforts and time isn’t wasted fixing bugs that users will never uncover.

Coordinating software distribution, beta usage, and defect tracking during OT can significantly impact scheduling and time-to-market. Often, a project cannot accept these costs. However, testing operational or system-level functionality alone can’t replace the value that OT provides in terms of the software quality perceived by users.

These two examples of testing vocabulary confusion are not unique. It isn’t difficult to find disparity or flat-out disagreement on testing vocabulary in testing literature.

As I’ve gained experience in testing software, I’ve found that new projects or colleagues frequently warrant a round of vocabulary diplomacy before beginning the work in earnest. This verbal exchange isn’t about whose definition is correct or who knows more. Also, it rarely makes sense to change a testing nomenclature already rooted in a project. However, you can establish a common ground, ensuring that best practices for quality assurance are executed, regardless of what they’re called. 🌐

John Steven is a consultant at Cigital with experience in consulting, distributed systems architecture, operating systems, and software-quality and security research. Contact him at Cigital, 21351 Ridgeway Cir., Ste. 400, Dulles VA, 20166; jsteven@cigital.com.