Robin F. Goldsmith, JD
robin@gopromanagement.com

"We don't need no stinkin' process!" You've probably heard some variant of this phrase, which evokes images of the banditos in "The Treasure of Sierra Madre," doesn't it?

When Development doesn't get you code to test on schedule, let alone tell you what the requirements are that the code is supposed to satisfy, you might possibly think of your Development colleagues as banditos, unwilling to play by the Quality Assurance rules prescribed for the good of all. Of course, that could never happen in your shop; but I'll bet you have your own local bandito stories.

So, what are we white hats in QA to do? I'm the first to encourage attending training and conferences, not only because I make a good part of my living there, but because I truly believe learning can help testers contribute more and find greater satisfaction by improving their effectiveness.

Unfortunately, testing gatherings too easily can become diverted into mainly being support groups seemingly devoted to commiserating with each other about how awful those banditos are. Gurus at the podium unwittingly feed into the milieu rehashing the "quality is important, and they ought to listen to us more and involve us better" sermon to the choir of the converted. While encouragement and positivism are essential, there are limits to the value and even real downside dangers of these well-intentioned messages.

One of the real risks of cheerleading without more is what I call the "Conference Attendee's Paradox." Consider this all too familiar scenario: Podium gurus implore us to go back and demand that the banditos involve us in requirements reviews so we can learn what we need to know about the requirements in order to test the delivered system and so we can spot requirements testability issues.

Revved up by the rhetoric, a tester returns from a conference, in fact talks his/her way into requirements reviews, and then after participating in one or two reviews summarily is dis-invited from further attendance, either literally or figuratively (such as being allowed to be present but ignored, or worse).

The well-intended seemingly wise advice from the podium gurus backfires, more than you might realize; and the gurus seem totally unaware of even the possibility their wisdom might be missing a nuance or two. Here are a couple of likely reasons.

First, just because I'm from QA/Testing doesn't mean I'm prepared to contribute. Requirements are mainly about content, so I need to know about the subject area. Frankly, often I don't know as much as I need to know about the business; and the more sets of requirements I get to review, the greater the chances are that I'm dealing with content I know too little about..

Second, education is not the purpose of a review. Although many organizations do view reviews are mainly educational, that focus diminishes the value of the review for accomplishing its real purpose—finding potential problems earlier when they can be fixed easier and cheaper. My guru and I may think I should be there to learn about the requirements; but the other folks probably expect me to be present mainly so I can find potential problems. *Before* going into that review, I need to invest some additional time getting the education I need so I can contribute productively during the review.

Third, there's a good chance that the gurus unintentionally have misled me about how I should participate in a review. The gurus probably emphasized that I should concentrate on spotting requirements that are not testable, which the testing community generally regards as the main requirements issue.

After all, I should be an expert at identifying testability issues; and by assuring requirements are testable, I'll be able to do my job of writing tests that demonstrate the delivered system meets its requirements. Moreover, since requirements which are not testable are likely to be misinterpreted by the developers, detecting untestable requirements will help prevent downstream problems.

Testability *is* important. Unfortunately, focusing almost entirely on testability can have some drawbacks that the testing gurus probably don't recognize. While it may not be fair, other folks probably don't see much value in tying up their time in reviews just so I can write better test cases; and the testability issues testers get all excited about sometimes strike others as trivial nitpicking, possibly with good reason.

When testers are mainly passive observers in reviews so they can learn what the requirements are, it's pretty likely that others don't perceive much value in the testers' presence and may not go out of their ways to continue it. If the testers are actively annoying people because they are perceived as tying up reviews with testability trivialities, it doesn't take long for the banditos to decide they don't need no stinkin' part of this testing process. In either situation, sooner or later, it's Adios, testers.

The conference attendee's paradox undercuts what seems on the surface to be very supportable good ideas for making testing more effective and more valuable to the organization. That's obviously a problem. A perhaps bigger problem is that we generally don't learn the right lessons from it.

We tend to blame the unfulfilling outcome on the other folks, the banditos, whom we declare obviously don't appreciate or care about quality. It gives us something to commiserate about when next we gather with other testers, who undoubtedly will be sharing similar tales of woe. What we usually don't see is how much we brought on ourselves a predictable undesirable outcome, which also unfortunately often turns out to have been our one bite at the early involvement apple.

Why don't we blame the testing gurus when their advice leads us into such frustration? We don't blame them because their advice was essentially right and more importantly agreed with our own beliefs. The problem stems from the fact that their actually not-so-good advice was grounded in a traditional *reactive* tester-centric mindset that we probably share with the gurus.

Proactive Testing™

If we want different results, we've got to do something different. I suggest we start by turning around our mindset and adopting a methodology I call Proactive Testing™. The methodology incorporates proven testing concepts and techniques, including testing/reviewing each development deliverable at the life cycle point it is delivered. However, Proactive Testing™ approaches these methods in more productive ways than traditional gurus usually are familiar with, including use of additional special techniques that help reveal numerous test conditions traditional reactive testing ordinarily overlooks.

The word "Proactive" is capitalized because it has a very special meaning that turns users, managers, and developers into advocates for testing. It certainly can include but is way more than generic uses of the term "proactive," such as merely encouraging good relationships, taking a developer to lunch, or initiating talk about how important quality and testing are.

"Proactive" with a capital "P" means anticipating, taking initiative, and providing WIIFMs. WIIFM stands for "What's In It For Me," where the Me is the other person whose behavior we want to influence. Too often Quality and Testing people get the equation backwards, essentially saying, "You do this because I think it's important." When we get Proactive, we find things that the other person really does think are important, usually things that save that person time, effort, and aggravation.

When we're Proactive, we make sure they get those WIIFMs; and we make sure they realize we've helped them get What's In It For Them. Instead of perceiving testing as an obstacle to their progress, which so often causes the user, manager, and developer banditos to feel the need to resist traditional reactive testing, they actively want Proactive Testing™ because they see that it helps them with what really matters to them.

For instance, whereas traditional testing typically uses one or two relatively weak techniques to review requirements, Proactive Testing™ applies more than 21 ways, including many which are much more powerful. Testability is only one of those 21+ ways; and as we said, testability often tends not to be a crowd favorite. Besides its tendency toward triviality, testability suffers from two other far more important but seldom explicitly recognized weaknesses. You see, testability has little to do with correctness. A requirement can be perfectly testable and perfectly wrong. Moreover, testability is totally irrelevant with regard to a requirement which has been overlooked.

People are much more appreciative of and interested in Proactive Testing™ reviews that also identify overlooked and incorrect requirements, especially when those requirements are theirs. The 21+ ways are described in my book, *Discovering REAL Business*

Requirements for Software Project Success, its two-day in-house seminar counterpart, *Defining and Managing User Requirements*, and in my live (public and in-house) and online one-day *Evaluating Business Requirements* seminar. For in-house clients, a two-day seminar version titled, *Testing Early in the Life Cycle--Reviewing Requirements and Designs*, also describes more than 15 ways to evaluate that system designs are accurate and complete.

Proactive Testing™ includes a number of additional powerful concepts and techniques which other of my various courses address. It's common for users to resist participating in User Acceptance Testing (UAT). Often UAT is a lot of extra work that doesn't catch many of the errors, and it's frustrating when errors that are caught aren't fixed because "it's too late." Such common UAT shortcomings are largely due to traditional reactive testing mindsets that mistakenly consider UAT a disempowering rubber-stamp proof-of-concept subset of system testing. In contrast, users do want to participate in Proactive User Acceptance Testing™. Its empowering Proactive approaches and methods build users' competence, confidence, and commitment.

By using powerful Proactive techniques early to plan and design their User Acceptance Tests for execution late, users both create a much more thorough set of tests and help catch requirements problems before they become bigger problems. Testers, analysts, managers, and users all can benefit from learning in my *Proactive User Acceptance Test* course how they actually can do more effective testing in the same or less time. WIIFM? You bet!

We've been focusing on the early life cycle aspects of Proactive Testing™ because these have high paybacks that often are not addressed adequately by traditional QA/Testing. However, Proactive Testing™ also dramatically improves what's usually the larger component of QA/Testing: planning, design, execution, and management of testing the delivered code.

Proactive Testing™ is an iterative risk-based approach which capitalizes on powerful special high-payback test planning and design techniques that can prevent the up to 75 percent of showstopper defects that traditional (including Agile) development and reactive testing ordinarily overlook. Such Proactive Master Test Planning Risk Analysis actually is spotting significant system design errors in a non-review manner. Users, managers, and developers easily appreciate and are grateful for finding these problems before they do turn into showstoppers.

Avoiding showstoppers are huge WIIFMs that turn banditos into Proactive Testing™ advocates. Participants get hands-on practice using these techniques in several of my courses which are available for in-house presentations, including my two-day *Proactive Testing™: Risk-Based Test Planning and Design* seminar, and my one-day *Risk-Based Testing* and *Proactive Testing™ Management* seminars.

Because Proactive Testing™ is anticipating rather than reacting to risks, higher risks can be tested not only more, but earlier, when they are easier to address. Moreover, Proactive Testing™ planning and design actually supercharge current buzzword techniques--such as continuous integrated testing, pair programming, and test-driven development--by more economically helping developers code more correctly in the first place and catch more of their own fewer remaining errors.

Proactive Testing™ applies powerful special techniques which identify numerous ordinarily-overlooked test conditions at three different levels: showstopper, feature, and test case. Risk-based prioritization enables maximizing productivity by continually refocusing on the higher risks at each level.

Furthermore, because Proactive Testing™ is efficiently anticipating, it is possible to increase reuse of test cases and especially test designs, which enables carrying out many more tests in the same amount of time. My public and in-house one-day *Developing Reusable Test Designs* seminar provides extensive hands-on practice identifying and reusing far more thorough than usual sets of test cases. These techniques are especially valuable for more economically and thoroughly populating automated test scripts and action word testing frameworks.

Finally, Proactive Testing™ provides a powerful structure for managing and improving testing, as described in three of my one-day courses which are available online as well as in public and in-house live formats: *Managing the Test Execution Process*, *Managing Testing Projects* (scheduled to be online March 2007), and *Test Process Measurement and Improvement* (online January 2007).

When users, managers, and developers perceive your Proactive Testing™ process as actually helping them get their WIIFMs, they become advocates for Proactive Testing™ and don't need to act like banditos. See you in an upcoming course!