# Good Practice Hunting

*by James Bach, james@satisfice.com*

*First published in Cutter IT Journal*
*February, 1999*

Imagine this conversation between a driving consultant and a client looking for best practices for his delivery truck company:

**Client:** "I want my drivers to drive at the right speed. In your expert opinion, what is the right speed?"

**Consultant:** "Hmm. I can't give you a specific number. It depends on a lot of factors."

**Client:** "Surely there's a speed that most good drivers generally drive. I don't need abstract driving theory, I just want to know what's the best practice out there."

**Consultant:** "Good drivers tailor their speed to the situation. There isn't any one speed that's best."

**Client:** "Oh, obviously we must hire good drivers. That goes without saying. All I really need from you is to tell me what our standard speed should be, based on what the best drivers do. Then our drivers will either use that speed or propose an alternative, as long as they justify their plan and follow it. We're ISO 9000 registered, you know."

**Consultant:** "The driving process requires different speeds at different times. Any given drive may involve speeds of 0 to 70 miles per hour. You can't know the speeds in advance, except very generally. The driver must make a situational judgment."

**Client:** "Wow, that sounds like it's up to the creativity of individual drivers. How will we get to Level 2 on the Driving Maturity Model? I don't need a consultant who says, 'it depends' and offers loosey-goosey guidelines; I need a concrete practice. Capers Jones gives concrete advice and hard numbers— why can't you?"

I know, this is a silly conversation. It's down-right ridiculous. No one who knows anything about driving would ever seek to settle on a single, standard "best speed." Yet in every *essential* respect, this dialogue is typical of how most of us, most of the time, talk about best practices in SQA and testing. We look for the one "right" answer, rather than considering the context. *We* includes me, too. Just yesterday, I caught myself, mid-sentence, arguing that a certain practice was "bad," even though I had not considered the context in which it was suggested. It's a seductive habit, but one that's well worth kicking.

The *goodness* of a practice is not an intrinsic attribute, of course. Rather, it emerges from the context of its use. I find it useful to think of that context in terms of capability, goals, and situation. Capability is how able we are to do what needs to be done. Goals are what we want to achieve. And situation is a catchall that represents the current state of the world as it relates to the practice.

The driving dialogue seems ridiculous only because, in that case, the roles of capability, goals, and situation are pretty obvious: a competent licensed driver will choose a reasonable speed; the goal is to deliver quickly without crashing or getting a ticket; situational factors that affect speed are commonly understood. Capability, goals, and situation are not nearly as clear in the case of software methodology. So, it's harder to determine the right course of action, and because it's harder we are all the more desperate to believe in inherently good or bad practices. We want something to hold onto.

## MYTHOLOGIES OF TESTING

Unfortunately, we aren't very good at observing and evaluating practices. So determining good practice is more often a process of *mythology*, not engineering or science. By that I mean our

analysis of practice is generally unsystematic, anecdotal, biased, history-bound, personality-driven, vague, exaggerated, and otherwise invites poetic license. And we generally don't question our myths, passed down as they are from Elders or Experts, even when they are disseminated outside of the context in which they were originally conceived and blessed.

Part of the problem is that it's hard to know what practices we are actually using. Does your organization perform unit testing? Are you sure? Unit testing is almost universally recommended and, according to more than one textbook on my bookshelf, is almost universally practiced. Well, that's interesting, because in my experience very few companies perform unit testing, and of those that do, there is a wide variation in the practice from developer to developer.

Part of this apparent paradox can be explained by looking at the definition of the term. I understand the notion of unit testing as the testing of individual modules, functions, or classes without regard to their integration with the rest of the system. The goal of unit testing is to find problems before integration, and to find those problems that are difficult to isolate on a subsystem or system level. The thing is, I've found that this definition is not the one in popular use. In my experience, unit testing more often means any test activity performed by a developer in the course of development—and most developers have no training in or passion for testing. It's become a minor hobby of mine to ask developers to describe what they do when they unit test. The typical answer is some variation of "I exercise the code and see if it works," which I've discovered could mean anything up to and including "I don't do anything."

The often startling difference between the *talk* and the actual *walk* is what I call a methodology gap. Unit testing stands out as a particularly stark example, but I find such gaps, to one degree or another, everywhere I look in typical software projects. Anyone who seeks to assess practices simply by asking practitioners if they're practicing them is likely to get a mythological and inaccurate picture of that organization.

Another part of the problem is that we don't have very good theoretical models of software engineering. Software engineering is generally treated as a process of efficiently creating software that correctly and dependably fulfills a specification. Testing, in that paradigm, is a process of generating the smallest set of tests that will reveal discrepancies between the software and its specification. That is certainly an interesting theoretical basis for testing, but I daresay it has not proven very helpful for the overwhelming majority of all the testing we do in this industry. Other branches of science offer models that might put our mythology on better ground. Economics, game theory, and decision theory offer insight into making tradeoff decisions. Principles of cognitive psychology and epistemology could help us understand how testers learn about and evaluate what they test. General systems theory offers ways to decompose and analyze the behavior of complex systems.

As a result of our incomplete theories and inadequate information, some of the most common and accepted wisdom about testing is at best misleading and at worst damaging:

### "It's important to repeat the same tests on each new build."

Not necessarily. This practice is usually justified by the frustration we feel when a problem is introduced into a product that could have been detected by an old test but wasn't because we never reran the old test. But my experience shows (caution: this, too, is mythology) that this problem is usually far smaller than the problem of not finding defects that were in the product all along because we were too busy rerunning old tests and didn't create enough new and different tests.

### "It's important to document all test cases and procedures."

Maybe, maybe not. Documenting test cases and procedures has at least two negative effects: (1) it tends to result in less overall testing, because of the time needed to create and maintain documentation, and (2) it tends to limit the variety of testing, because having documented the tests, testers tend to feel obligated to execute only those tests. In the absence of a pressing need for external accountability or a pressing need to share exact test cases with lots of other people, it is probably better, in my opinion, not to document specific tests. Document just enough to remind you to cover what you need to cover and report what you need to report.

**"It's important for testing to be involved early in the test cycle."**

That might be a complete waste of time. Most testers want to be involved early in the cycle, but few know what to do when they find themselves there. I can think of a lot of things to do early in a test project, but they require the skills of a technically savvy senior tester or test manager. An unsophisticated or indiscreet tester will only alienate the development staff.

**"It's important to create tests based on specifications well before it's time to execute them."**

Not so fast. Specifications are notoriously and almost universally inadequate for the generation of actual tests. Attempts to do that usually result in a lot of mostly useless documentation. It's certainly useful to examine specifications for testability and consult with the developers about improving testability, but that requires unusual skill. Reporting defects in the specification may be useful, too, but that requires unusual rapport and diplomacy.

Each of these commonly suggested practices is good to do—in certain situations, when you're trying to accomplish certain goals, and you have a staff capable of performing them. They are not, however, inherently good practices.

I'm not against mythology. The kind of research that might provide a comprehensive scientific foundation for methodology is expensive and, for the most part, infeasible. So, we're stuck with mythology—this article is mythology—but let's be open about it. If a testing practice seems to make sense, do it. But beware of common wisdom.

## COMMUNITIES OF PRACTICE

My concerns about the situational goodness of practices and the difficulty of observing them strike many practical people as academic, and that's understandable— the fact is that in software development, we can think of many practices that most of us will agree are generally good or generally bad. Some companies have reported tremendous success with certain practices.

Yet the reason any practice seems inherently good or bad is that we make assumptions about capability, goals, and situation. Though we are often not even aware of those assumptions, they create a foundation for evaluating and evolving practices *as if* they were standalone entities. Within a community of people who have a bedrock of shared assumptions (something I call a *community of practice*), it is indeed meaningful to say that one practice is good and another is bad. "Most of us" agree because we belong to the same community, and we aren't even in dialog with practitioners in other communities.

What made it possible for other fields of engineering to produce standard handbooks of good practices and certification tests for engineers is that each domain of traditional engineering represents a sufficiently cohesive community of practice and has evolved a theoretical and technological foundation to support the evaluation of its practices. These communities can reasonably account for the variables of situation, goals, and capability. The problem for the software industry is that we consist of umpteen overlapping and fragmented communities, yet our discussions of practice don't account for those communities. In the testing world, I see at least the following broad communities of practice:

- **Regulated:** obligated to prove compliance with external standards
- **High reliability:** spares no effort or expense to produce a high-reliability product
- **Academic:** explores theory at the expense of practical applicability
- **Contract-driven:** obligated to fulfill a specific contract with a specific customer
- **Market-driven:** aims to fulfill a general market need in a competitive environment
- **Embedded:** provides software only as an adjunct to a hardware system
- **IS:** provides business-critical technology for internal users

Cutting across these communities are others that are specific to application domains (such as medical information systems, process control systems, or desktop business applications) or those that are specific to particular technologies (such as relational databases, Internet, or Java).

## WE'RE ALL METHODOLOGISTS

If testing practices are based on mythology, and that mythology varies from community to community, where does that leave us? It leaves us to our own devices, more or less. To the extent that we pursue excellence as testers and test managers, each of us must become our own methodologist. Therein lies another problem. If there's one thing I know for absolute certain as a full-time methodologist, it's that most people have no patience for discussing process. Nevertheless, I don't see how it's possible to do truly and demonstrably excellent testing in this business unless you take control and ownership of testing mythology.

Doing methodology means observing and puzzling over the forces that influence how we do our work and struggling to articulate models, methods, and heuristics that help us grasp and communicate the essence of the testing craft. One humble and helpful tool I can offer is the following list of questions. When I am considering adopting a testing practice— for example, using a new test plan template or test tool— I walk through this list:

1. What objectives are served by this practice? What pain will it resolve?

2. Are those important objectives? Important to whom?

3. In what way are those objectives already served by some other means?

4. What would a highly successful implementation of this practice be worth?

5. How much energy will be required to make it happen? Is there a simpler, cheaper solution?

6. What are the prerequisites for adopting this practice (e.g., special training, methods, or tools)?

7. How will this practice disturb or interact with existing practices or processes?

8. What problems or risks will this practice create?

9. How will we know that the practice is helping? How will we assure its quality?

10. If it isn't helping, what will we do then?

11. How much of this practice will be enough, or too much? Can a little of it make a big impact?

12. What alternatives are there to this practice? What if we do nothing?

13. What simple, achievable, self-contained step can be taken toward the new practice?

I recently had a brief argument with my friend and colleague Johanna Rothman on the subject of doing regression testing on specific defects that had already been fixed. She thought it was very important to recheck fixed bugs on a regular basis; I thought it was a waste of time. Finally I rolled out some data to prove my point (mythology alert: this data is from a single project, several years ago), and I told her that on a population of 2,000 defects fixed over a six-month period, I had measured only a 2% recurrence rate. Aha, but Johanna had data of her own that showed a recurrence of 40%! When she said that, I felt incredulous for a moment, then almost said, "You're crazy, Johanna. That number has got to be wrong." But I didn't say it. Another, wiser thought popped up, and I asked instead, "Johanna, how can we account for the difference in our data?" Well, we worked it out, and sure enough, I finally had to admit that for her particular technology, rechecking certain classes of old defects should be an essential part of the test strategy. Situational practice wins again.

And that's the thought I want to leave you with. The next time you feel the urge to pass ultimate judgment on the goodness or badness of a testing practice, pause a moment. *It all depends.*

*James Bach is an independent SQA consultant, speaker, and writer. He can be reached at:*

*j.bach@computer.org*
*http://www.jamesbach.com*