Author Contact: james@satisfice.com

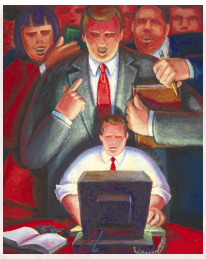
A Framework for Good Enough Testing

James Bach, Reliable Software Technologies

After a year in Sunnyvale, California, doing test management, I'm pulling up stakes again. I'm off to Virginia, where I've been hired by Reliable Software Technologies, a company known for tools and consulting dedicated to achieving very high quality software. I'll spend most of my time in a consulting role, which is a very different reality from software management. But it will be an interesting one, nonetheless, especially since I've never worked in an environment that put reliability ahead of marketability, or even abreast of it.

I'm known for my Good Enough quality model ("Good Enough Quality: Beyond the Buzzword," *Computer*, Aug. 1997, pp. 96-98), so it may seem strange that RST recruited me. At first, I thought CEO Jeff Payne had let his *Computer* subscription lapse, and thus hadn't encountered my philosophy of quality. But as it turns out, RST believes that a very high reliability standard is—for them—just part of good enough quality.

The same thinking about quality applies no matter where we place ourselves on the quality scale: At every point in the life cycle, we must compare the present quality of the product against the cost and value of further improvement.



I want to distinguish the Good Enough approach from the approach that says that we should keep testing until Management pries the product from my cold, dead fingers.

WHAT MAKES SENSE WHEN?

The challenge I face in moving to RST is to be able to articulate, and perhaps quantify, why certain demanding test techniques might make sense for a mission- or life-critical project, and, if so, how much testing is needed. Everybody knows that exhaustive testing is impossible, in both principle and practice. But sufficient testing might be possible, depending on the circumstances.

I want to distinguish the Good Enough approach from the approach that says we should do every kind of test we can think of and fix every bug we find and that I should keep testing until Management pries the product from my cold, dead fingers. As far as I can tell, this is an approach that never knows when to say when and doesn't care.

There has been a lot of criticism of Good Enough from this "exhaustionist" perspective. But exhaustionism is irresponsible: The test manager who refuses to face the fact that exhaustive testing is impossible chooses instead to seek an impossible level of testing. This is a purely political refuge: When Management ships the product over the inevitable objections, the test manager can blame Management for every bug found in the field: "I *told* them it needed more testing!"

Within the testing industry, we're really struggling with how to know when to say when. For the last few years, Cem Kaner (*Testing Computer Software*, 2nd ed., International Thompson Computer Press, 1993; and *Bad Software: What to Do When Software Fails*, John Wiley & Sons, New York, 1998), Brian Marick (*The Craft of Software Testing*, Prentice Hall, Upper Saddle River, N.J., 1995), and I have been discussing and debating it.

We are meeting and working with a growing community of testers, test managers, and consultants who consider themselves Good Enough proponents, and I believe the time is right to propose specific models of Good Enough testing.

GOOD ENOUGH TESTING DEFINED

In any situation, Good Enough testing asks, "How do I know if I'm doing, or have done, enough of the right type of testing?" Unfortunately, there is no objective or rigorous calculus for answering this question, but we can identify what to consider in attempting to answer it. We can begin to build at least a heuristic framework around the problem. In fact, my general model of Good Enough quality could apply to Good Enough testing as well. I'm borrowing some elements from it, but here I'm proposing a more specific framework.

As a first step, I'll define the term:

> Good Enough testing is the process of developing a sufficient assessment of quality, at a reasonable cost, to enable wise and timely decisions to be made concerning the product.

Let me unpack this definition. I'm asserting that, at a high level, it's useful to think about the value of testing as a dynamic with four parts:

- *Assessment of product quality* (How accurate and complete is it?).
- *Cost of testing* (How reasonable is it? Is it within project constraints? Is there a good return on investment, such as the extent of information gained per test?).
- *Decisions* (How well does the assessment serve the project and the business?).
- *Timing of all the above* (Is it soon enough to be useful?).

Testing also supports business processes beyond decision-making. But for this short column, I am lumping under "decisions" the uses any testing client would have for test results, such as creating accurate marketing materials or improving our ability to provide technical support.

In general, testing is better if the assessment of quality is more accurate, the cost of testing is lower, the basis for making decisions is better, and the time frames are shorter. Perfect testing would be to instantly and effortlessly give the right information to allow any part of the business to make any necessary decision concerning the product.

By this definition, perfect testing is easy to achieve in some screwed-up projects. One example comes from my colleague Doug Hoffman, who was once in a situation where he was told that no testing he could do would affect the decision to ship. So he pronounced the testing complete.

In a different situation, perhaps continuing the testing would have offered a benefit for technical support or provided a basis for some other type of corporate decision. The point is that when the testing is not coupled with a decision to be made and is not providing data for future use, that testing has no purpose.

Another all-too-common situation arises when an organization or regulatory authority *requires* certain testing or certain test products even if they contribute poorly or not at all to the project. Although I can see their political necessity, such test products have little to do with what I'm proposing. Good Enough testing is about conscious and purposeful testing, not superstition and ritual. Most test plans that I've seen could be torn up and thrown away with absolutely no effect on the test project or any stakeholders.

> At every point in the life cycle, we must compare the present quality of the product against the cost and value of further improvement.

In many cases, test plans are written because somebody said, "The rule book says we're supposed to have one of those." I wrote a couple of those myself, years ago, so I don't mean to sound as if I've never been seduced by the Dark Side of the Force. But as evolving professionals, we should be striving to contribute more value and less clutter to our projects.

COMPONENTS OF ASSESSMENT

Seriously addressing the Good Enough testing question involves first assessing the four parts of the definition and then deciding if they are good enough as a whole, or whether it is worth improving them by improving the test process. You can apply this analysis to any test methodology I can think of.

1. Assess product quality

- How are we assessing and reporting the quality of the product?
- Are we sure our assessment of quality is justified by our observations?
- Are we aware of the stated and implied requirements of the product when we need to know them?
- How quickly are we finding out about important problems in the product after they are created?
- Are our tests covering the aspects of the product we need to cover?
- Are we using a sufficient variety of test techniques or sources of information about quality to eliminate gaps in our test coverage?
- What is the likelihood that the product could have important problems we don't know about?
- What problems are reported through means other than our test process, that our testing should have found first?

2. Evaluate the cost of testing

- How much does the testing cost? How much can we afford?
- How can we eliminate unnecessary redundancy in our test coverage?
- What makes it difficult (and thus costly) to perform testing?
- How might the product be made more testable?
- Are there tools or techniques that might make the process more efficient or productive?
- Would testing be less expensive overall if we had started sooner or waited until later?

3. Check how well testing supports decision-making

- Is the test process aware of the kinds of decisions management, developers, or other clients need to make?
- Is the test process focused on potential product and project risks?
- Is the test process tied to processes of change control and project management?
- Are test reports delivered in a timely manner?
- Are test reports communicated in a comprehensible format?
- Is the test process communicated, as well as test results? Are we reporting the basis for our assessment and our confidence in it?
- Is the test process serving the needs of technical support, publications, marketing, or any other business process that should use the quality assessment?

4. What about timing?

Every aspect of the three other parts of the model is time-driven. That's the problem: We never have enough time to do everything, so everything we do is a race against the clock.

BRINGING IT TOGETHER

In the next part of the assessment, consider your answers and ask, "How good is our testing?"

How good is our testing?

- With respect to the preceding questions, are there any pressing problems with the test process?
- Is our test process sufficient to alert management if the product quality is less than they want to achieve?
- Are any classes of potential problems intolerable, and, if so, are we confident that our test process will locate all such problems?

> The test manager who refuses to face the fact that exhaustive testing is impossible chooses instead to seek an impossible level of testing.

Is it worth improving?

- What strategies can we use to improve testing?
- How able are we to implement those strategies? Do we know how?
- How much cost or trouble will it be to improve testing? Is that the best use of resources?
- Can we get along for now and improve later? Can we achieve improvement in an acceptable time frame?
- How might improvement backfire and introduce bugs, hurt morale, or starve other projects, for example?
- What specifically should we improve? Are there any side benefits (such as better morale) to improving it?
- Will improvement make a noticeable difference?

I like models that can apply to any type of software project situation. I'll wager that this set of questions is worth considering, whether the project is life-critical or merely market-critical.

However, it's often easier to answer these questions with hindsight, by comparing the information revealed through testing to information revealed by other means, such as customer experience. In the market-driven software world, most software goes through many iterations of development and release, so hindsight can be enough.

The iterations provide a means to assess and improve the test process. In the mission- and life-critical worlds, though, the challenge is to know that testing is good enough while doing it the first time. We will be struggling with that challenge for years to come.

My main concern in all this, apart from being useful at RST, is helping the software testing profession work itself out of politics, subjectivity, and defensiveness and instead apply structure and rationality to a difficult, multidimensional set of problems.

When enough testers choose honestly to face the trade-offs that confront us, we'll have the foundation for a Good Enough profession. ❖

James Bach is a principal SQA practitioner at Reliable Software Technologies. Contact him at j.bach@computer.org.

COMPUTER
Innovative technology for computer professionals

Circulation: *Computer* (ISSN 0018-9162) is published monthly by the IEEE Computer Society. IEEE Headquarters, 345 East 47th St., New York, NY 10017-2394; IEEE Computer Society Publications Office, 10662 Los Vaqueros Circle, PO Box 3014, Los Alamitos, CA 90720-1314; voice (714) 821-8380; fax (714) 821-4010; IEEE Computer Society Headquarters,1730 Massachusetts Ave. NW, Washington, DC 20036-1903. Annual subscription included in society member dues. Nonmember subscription rate available upon request. Single-copy prices: members $10.00; nonmembers $20.00. This magazine is also available in microfiche form.

Postmaster: Send undelivered copies and address changes to *Computer*, IEEE Service Center, 445 Hoes Lane, Piscataway, NJ 08855. Periodicals Postage Paid at New York, New York, and at additional mailing offices. Canadian GST #125634188. Canada Post International Publications Mail Product (Canadian Distribution) Sales Agreement Number 0487910. Printed in USA.

Editorial: Unless otherwise stated, bylined articles, as well as product and service descriptions, reflect the author's or firm's opinion. Inclusion in *Computer* does not necessarily constitute endorsement by the IEEE or the Computer Society. All submissions are subject to editing for style, clarity, and space.