Editor: James Bach, ST Labs Inc., 3535 128th Ave., SE, 3rd Fl., Bellevue, WA 98006; jamesb@stlabs.com

Good Enough Quality: Beyond the Buzzword

James Bach, ST Labs Inc.

I kicked off this department in February by examining the gap between what we say and what we do in software projects ("The Hard Road From Methods to Practice," Feb. 1997, pp. 129-130). In any given project, our publicly declared methodologies often bear little resemblance to our actual practices. In this article, I'd like to examine one actual practice that is finally emerging as a describable method: Good Enough Software.

A few months ago, at a software development conference, I heard Larry Constantine declare that he doesn't believe in the idea of good enough software quality. Constantine's disdain was so clear, I asked him why he even bothered to dignify the idea with a critique. His reply: "Because everybody's talking about it."

I too hear talk about Good Enough, but it comes almost exclusively from people on software projects, rather than from people who write or consult about software processes. It's hard to find much of anything about Good Enough in software engineering literature, or at the conferences. There was an article in the *Communications of the ACM* a few years ago (W. Robert Collins et al., "How Good Is Good Enough?: An Ethical Analysis of Software Construction and Use," Jan. 1994, pp. 81-91). I've written one obscure article on the subject ("The Challenge of Good Enough Software," *American Programmer*, Oct. 1995) and spoken about it here and there. Ed Yourdon has written about it, too, mostly in reference to my work and to his experiences interviewing developers at Microsoft. Although the basic ideas can be found in the work of Gerald Weinberg, Robert Glass, Fred Brooks, and many others, I believe it was Yourdon who first elevated the notion from plain old English phrase to full-fledged technical buzzword.

A new buzzword! Like a new island rising out of the Pacific, another long-neglected bit of software reality is bursting into the canon of software engineering methodology. But as with any volcanic event, this new idea is subject to chaos and confusion.

CHAOS AND CONFUSION

For example, Capers Jones devoted five pages of his recent book (*Software Quality: Analysis and Guidelines for Success,* International Thomson Press, 1993) to debunking what he called the "good enough quality fallacy." As Jones put it, "from the observed fact that most commercial software contains bugs at delivery, the 'good enough' enthusiasts have formed the hypothesis that leaving bugs is a deliberate and even clever strategy on the part of commercial software houses, which might advantageously be imitated by other software developers." He also stated, "Companies such as Microsoft do not deliberately ship software that contains bugs."

I believe this reveals a poor understanding of what I consider to be the Good Enough approach. And how do I know what Good Enough Software is? Because Good Enough Software—as I define it—is just a refinement of what I experienced virtually every day for 12 years as a developer and test manager in major commercial software companies. My model is not based on some armchair hypothesis—it describes my real experience. Furthermore, my company, ST Labs, has run testing projects for hundreds of companies, and we find that it is *routine* for our clients to ship with known bugs.

Cem Kaner, in *Testing Computer Software* (International Thomson Press, 1993)—standard issue to Microsoft's testers—also refers to bug fix deferral as a routine practice. I know this is true at Microsoft because my company runs a dedicated test lab there. I speak and teach there on a regular basis. My brother is even a test lead there. Take it from me, Microsoft begins every project with the certain knowledge that they will choose to ship with known bugs. This strategy works for Microsoft because it knows how to ship with the *right* bugs.

Take it from me, Microsoft begins every project with the certain knowledge that they will choose to ship with known bugs.

GOOD ENOUGH AS A PARADIGM

Confusion about Good Enough Software is understandable and forgiveable, since no one has published an actual detailed description of what Good Enough means. Jones seems to define it as the practice of deliberately leaving bugs in the code so as to shorten the schedule. I've heard other people define it as providing the minimum quality that you can get away with.

A Framework for Good Enough Quality

Taken together, these four GEQ factors and six GEQ perspectives comprise a robust set of reminders that can help frame a conversation or make a convincing case about shipping a product, improving it, or implementing some better practice. For instance, when someone tells me that "good enough is not good enough," I remember the stakeholder and critical purpose perspectives and translate that apparently paradoxical statement into something I can question, such as "good enough *for you* is not good enough *for me*" or "good enough to *survive* is not good enough to *succeed*." Then the dialogue becomes one of examining whose values matter or what purpose we are really trying to achieve.

GEQ Factors

This four-part process expands upon the GEQ definition. It's not a rigorous formula, by any means. These factors are designed to help busy, stressed-out software people remember what to think about when weighing product quality.

1. Assess the benefits of the product.

- **Identification.** What are the known benefits or potential benefits for stakeholders of the product?
- **Likelihood.** Assuming the product works as designed, how likely are stakeholders to realize each benefit?
- **Impact.** How desirable is each benefit to stakeholders?
- **Individual criticality.** Which benefits, all by themselves, are completely indispensable?
- **Overall benefit.** Taken as a whole, and assuming no problems, are there sufficient benefits for stakeholders?

2. Assess the problems of the product.

- **Identification.** What are the problems or potential problems for stakeholders of the product?
- **Likelihood.** How likely are stakeholders to experience each problem?
- **Impact.** How damaging is each problem to stakeholders? Are there workarounds?
- **Individual criticality.** Which problems, all by themselves, are completely unacceptable?
- **Overall problems.** How do all the problems add up? Are there too many noncritical problems?

3. Assess product quality.

- **Overall quality.** With respect to the GEQ perspective, do benefits appear to outweigh problems?
- **Margin of safety/excellence.** By how much do we need or want benefits to outweigh problems?

4. Assess the logistics of improving the product (or holding out for something better).

- **Strategies.** What strategies can we use to improve the product?
- **Capabilities.** How able are we to implement those strategies? Do we know how?
- **Costs.** How much cost or trouble will improvement entail? Is that the best use of resources?
- **Schedule.** Can we ship now and improve later? Can we achieve improvement in an acceptable time frame?
- **Benefits.** How specifically will it improve? Are there any side benefits to improving it (for example, better morale)?
- **Problems.** How might improvement backfire (for example, introduce bugs, hurt morale, starve other projects)?

GEQ Perspectives

The GEQ factors just listed are necessary but not sufficient. In order to perform a responsible assessment, you must also examine each of the factors from six vital perspectives.

1. **Stakeholders.** Whose opinion about quality matters? (For example, project team, customers, trade press, courts of law.)
2. **Critical purpose.** What do we have to achieve? (Is it immediate survival, profitability, market share, market growth, customer satisfaction?)
3. **Time frame.** What is the time-sensitivity of quality perception? (Is it immediate, near-term, long-term, after critical events?)
4. **Alternatives.** How does this product compare to alternatives, such as competing products, services, or solutions?
5. **Consequences of failure.** What if quality is a bit worse than good enough? Do we need a contingency plan?
6. **Quality of assessment.** How confident are we in our assessment? Is it good enough?

One reason why Good Enough is not better described may be that it is so much a part of our experience that it seems too obvious to mention. Only when contrasted with idealistic, normative models of software engineering does Good Enough stand out as a separate paradigm—a different pattern of assumptions about the way the world works. Methods for achieving Good Enough quality are of interest mainly within the larger context of the Good Enough paradigm.

Here are some of the basic assumptions that I believe are part of the Good Enough paradigm:

- We are obliged to cope with a world full of complexity, unknowns, limitations, mistakes, and general imperfection.
- People are by far the most variable and vital components of software projects.
- Everything has a cost, and what we want always exceeds what we can afford.
- Quality is ultimately situational and subjective.
- To achieve excellence in something as complex as software, we have to solve a lot of difficult problems, make a lot of trade-offs, and resolve con-

tradictory values. Excellence does not come easily or mechanically.
- Software engineering methods are useful to the extent that they are designed with these assumptions in mind.

Bear in mind that the real essence of Good Enough lies in the minds of practitioners, not in any practice. The paradigm is one of learning on the job, learning from failure, coping with complexity, and coping with humanity. It encourages healthy skepticism by building in the idea that benefits always come with problems. Our task is not to blindly eliminate all problems, but to understand the problems and benefits of a situation well enough to eliminate (or prevent) the *right* problems and also deliver the *right* benefits.

As a consultant, I see the Good Enough approach mostly as a way to drive ongoing improvement, whereby we approach excellence by progressively achieving, challenging, and raising our standard of Good Enough, as opposed to driving toward some abstract, apocryphal metric like six sigma, or a defect removal ratio of 99 percent, or a CMM level of 4. It's also useful as a Rosetta stone that helps quality specialists discuss quality with people in other specialties.

GOOD ENOUGH QUALITY ANALYSIS FRAMEWORK

There are a few of us in the industry who are quietly toiling away to develop and teach this as a discipline. Here I propose a framework for evaluating Good Enough Quality (GEQ). Let's start with a definition of what's GEQ. Here's my whack at it. To claim that any given thing is Good Enough is to agree with all of the following propositions:

- It has sufficient benefits.
- It has no critical problems.
- The benefits sufficiently outweigh the problems.
- In the present situation, and all things considered, further improvement would be more harmful than helpful.

Each point is critical. If any one of them is not satisfied, then the product, although perhaps good, cannot be good enough.

The first two seem fairly obvious, but note that they are not exact opposites. The complete absence of problems cannot guarantee infinite benefits, nor can infinite benefits guarantee the absence of problems. Benefits and problems do offset each other, but it's important to consider the product from both perspectives.

The third proposition reminds us that benefits must not merely outweigh problems, they must do so to a sufficient degree. It also reminds us that even in the absence of any individual critical problem, there may be a pattern of noncritical problems that essentially negate the benefits of the product.

The fourth proposition introduces the important matter of logistics and side effects. If high quality is too expensive to achieve, or if achieving it would cause other unacceptable problems, then we either must accept lower quality as being good enough or we must accept that a good enough product is beyond our reach.

Good Enough has nothing to do with mediocrity. It has to do with rational choices, as opposed to compulsive behavior.

These propositions can be expanded into a framework of GEQ factors that support a process of structured reasoning and dialog about quality, as summarized in the sidebar "A Framework for Good Enough Quality."

RATIONAL, NOT COMPULSIVE

I hope the framework makes it clear that Good Enough has nothing to do with mediocrity. It has to do with rational choices, as opposed to compulsive behavior. If something really is good enough, in terms of this framework, then further improvement means making an investment that has an inadequate return. When we find ourselves in that situation, we should look for hidden compulsions.

Is there anything new with the Good Enough approach? Not really. Maybe just its name. Henry Petroski has been writing about the role of failure in successful design for years. Herbert Simon coined the term "satisficing" in his book, *The Sciences of the Artificial* (MIT Press, 1992). Two excellent books on Good Enough software engineering (although neither book bills itself as such) are Gerald Weinberg's *Rethinking Systems Analysis and Design* (Dorset House, 1988), and Robert Glass' *Software Creativity* (Prentice Hall, 1995). A few years ago, I couldn't find any books about software risk management; now there are more than a dozen in print.

The big new force that is propelling the Good Enough idea is the explosion of market-driven software. With a passion roughly proportional to the price of Microsoft stock, companies are looking for the shortest path to better software, faster, and cheaper. They are willing to take risks, and they have little patience for the traditional moralistic arguments in favor of so-called good practices.

Much of the traditional lore of software project management seems irrelevant or stilted when applied to market-driven projects.

It's time that we developed approaches and methodologies that apply to the whole craft, not just to space missions, medical devices, or academic experiments. Good Enough is a model that encompasses high-reliability products as well as high-entertainment products. Whether you call the idea Good Enough, or choose another buzzword like economical, pragmatic, or utilitarian, the basic idea remains the same: Our behavior should be guided by reason, not compulsion.

Beyond the notion of "best practices" is a more fundamental idea: best thinking. As the Good Enough idea continues to emerge, the quality of our thinking, rather than our conformance to formalities, will become the issue. Formalities, and the authority behind them, will be reexamined. No wonder so many authorities consider Good Enough to be a dangerous idea. ❖