Edward Yourdon

WHEN GOOD ENOUGH SOFTWARE IS BEST

How to get people and technology to work together.

There's a sign that my printer displays prominently on his wall: "You can get it fast; you can have it cheap; you can get it right. Pick two!" That same sign could be displayed on the wall of every software-development organization. And yet most of our customers want all three. Ed Yourdon tackles that dilemma in this issue's column. He contends that we don't rationally establish proper balance among the critical project parameters: cost, schedule, staffing, functionality, and quality. Our customers want us to optimize all these parameters, even when this is clearly impossible. The purists among you may find Ed's comments grating. But I suspect that those of you who've been bloodied in the project wars will find wisdom in his words.

— Roger Pressman

IN THE BEST OF ALL POSSIBLE WORLDS, our users would like us to develop software instantly, at no cost, and with no defects. But that's not possible in today's world. In more and more application domains, we've been forced to accept that the reengineering slogan of "faster, cheaper, better" really means "fast enough, cheap enough, good enough."

In the past few months, the concept of "good-enough" software has been getting a lot of attention: the uproar over the Pentium bug suggests that it was deemed *not* good enough, while the surprisingly numerous defects that are publicly acknowledged in popular shrinkwrapped software products — word processors, spreadsheets, tax calculators, and PC operating systems — suggest that those products *are* good enough.

The concept of good-enough software is beginning to challenge some of our basic assumptions about software development, and I believe it will fundamentally change the way we manage software development. Some purists — especially veterans who tend to read magazines like this one, and who regard themselves as professionals — may express horror at this "flight from quality." But I think a good-enough approach will lead to more rational software-development projects and a more rational way of negotiating with our customers and managers on what constitutes success.

IT MAY SHOCK PURISTS, BUT THIS MAY BE A MORE RATIONAL WAY TO NEGOTIATE SUCCESS.

MISAPPLIED IDEAL. In the past, we often negotiated critical success factors *once*, at the beginning of the project, then tried to optimize a few other parameters the customer was often unaware of. For example, functionality, schedule, budget, and staff resources typically were negotiated in terms of political constraints: We were told to deliver a software system with a certain amount of (often ambiguous, misunderstood, and poorly documented) functionality within a (hysterically optimistic) schedule and (imposed by fiat) budget, and with a (relatively fixed) staff of developers. Within those constraints, the developers often tried to optimize such features as maintainability, portability, reliability, and efficiency. Thus, the battle cry for many projects was "We'll deliver high-quality, bug-free software on time, within budget!"

For an important class of software projects, that battle cry is still relevant — obviously, nobody wants to fly on an airplane whose guidance-control software has as many bugs as the word processor they can buy at their local computer store. And nobody wants their telephone or automated teller to malfunction as often as their PC.

But for another class of software projects — a class that is now arguably far larger than the mission-critical class — rapid delivery to the customer is sometimes more important than number of defects. In other situations, "feature richness" may be the most important factor; in still others, cost may be the only thing the user cares about.

WHEN BEST ISN'T. The shift in expectations we're experiencing stems from information technology's transition into a consumer commodity: unit costs are low and everyone can afford it. In the past, most of us worked on proprietary, one-of-a-kind systems, developed according to schedules measured in years and funded by budgets measured in millions. Some of us are still

Editor:
Roger Pressman
R. Pressman & Associates
620 E. Slope Dr.
Orange, CT 06477
rsp0547@aol.com

employed by organizations that want custom systems — but schedules and budgets have shrunk considerably. And our customers will often point out that they can achieve almost the same results by jury-rigging a combination of Microsoft Word, Lotus Notes, and Borland Quattro, which they can obtain from a discount mail-order catalog. Shrinkwrapped software may be clumsy and limited in its functionality, our customers tell us, but it's cheap and they can put it into service tomorrow morning.

LATE IS NEVER BETTER. That customers now view software as a commodity has also introduced an inertia we must cope with, especially in the consumer desktop market. It goes like this. Suppose the time has come to acquire a word processor. You have a choice of products A, B, and C. Product A costs $500 and comes with a money-back guarantee; product B costs $100 and comes with a long disclaimer that basically says "caveat emptor." Product C costs only $50, has twice as many features as either A or B, and its developers are so confident of its quality they're bragging about a double-your-money-back guarantee. The only problem with product C is that it's vaporware and (despite glowing reviews in all the trade magazines) won't be available for six months. Assuming you need a word processor *now*, you will probably make a rational choice between A and B based on your assessment of the importance of cost versus defects.

IT IS THE CUSTOMER WHO DECIDES THE PROPER BALANCE OF PARAMETERS.

But now suppose you already have a word processor, perhaps B, and you've been using it for a year. Some of its features are slightly annoying, but it's adequate for your mundane word-processing tasks. B's quality isn't all that great: it crashes once a day, and you've become accustomed to saving your documents every 15 minutes.

Now vendor C finally delivers its product and it really *does* cost only $50 and it really *does* have a level of quality 10 times higher than your existing product. Would you switch? Maybe — but maybe not. What if product C required you to convert all your existing word-processing documents to a different format? What if it required you to switch to a different operating system? You might well conclude that product B was good enough.

In this case, the project manager for product B has outsmarted the project manager for product C, even though C's manager pursued a set of goals that all software professionals would admire.

DO THE MATH. Software project managers today must be aware that *each* parameter — cost, schedule, staffing, functionality, and quality — is potentially critical. It is the customer — be they an end user for an in-house system or the marketing department for a software company — who decides what the proper balance is. It's also crucial to remember that the balance among parameters is dynamic and may need to be readjusted daily. After all, the business environment is likely to change in a dramatic, unpredictable way — and this can easily change the customer's perception of the importance of schedule, cost, and so on.

Intelligent customers, especially those who have survived today's tumultuous business environment, know trade-offs must be made and priorities balanced. But customers are often naive about the details. For example, it may not occur to them that defects (aka "bugs") are a parameter we must consciously plan for, and for which we must trade off other parameters. And of course customers may not want to make the cold-blooded, rational, calculated decisions about those trade-offs. Although immensely frustrating to developers, it's understandable that customers demand a software system in half the time, at half the cost, with twice the functionality and half as many defects as the developers believe technologically possible. They don't know any better.

What does this mean for the project manager? If we can assume for the moment that we're dealing with rational customers, and that a rational negotiation can determine the criteria for project success, then it is incumbent upon the manager to be as forthright and detailed as possible about *all* the relevant success criteria. Thus, instead of just *assuming* that the customer requires zero-defect quality, the project manager should say something like, "Our standard approach for developing the software you've described will require X number of people and Y units of time with a cost of Z dollars; we'll deliver P units of functionality with a defect level of Q bugs per function point."

Chances are that the proposed combination of X, Y, Z, P, and Q will *not* be acceptable to the customer, whose likely response might be "You can't have Y units of time, we need the software in half that time." Or a less rational customer might respond, "We want twice the functionality you proposed, but you can only have half as many people, half the time, and half the budget." The response then is that this is possible, assuming that the customer completely relaxes the constraint on the number of defects (most likely an unrealistic assumption). After all, I can deliver an infinite amount of software, with an infinite amount of functionality, in zero time — if it doesn't have to work. An even less rational customer might constrain *all* the parameters to some demonstrably unachievable level! It *is* perfectly rational for our customers to challenge our proposal for X, Y, Z, P, and Q — particularly if we can get them to focus their attention on one parameter at a time. If the user wants the software in half the time, then it's incumbent on us to provide a counterproposal that shows the effect such a change will have upon one or more of the other parameters.

Some 20 years ago Fred Brooks reminded us (*The Mythical Man-Month*, Addison-Wesley, 1975) that time and staff resources are not interchangeable in a linear relationship. If we reduce the project schedule by half it will *more* than double the required staff. Or we can cut the schedule in half, keep the staff con-

stant, and increase the cost in a nonlinear fashion (by having the constant-level staff work extraordinary levels of overtime).

NEGOTIATING A SUCCESSFUL PROJECT. The mathematics of the relationships between *X*, *Y*, *Z*, *P*, and *Q* are something we don't know enough about at our present level of software engineering. Larry Putnam and Ware Myers have explored this in their book, *Measures for Excellence — Reliable Software on Time, Within Budget* (Prentice-Hall, 1992), but much more work is necessary.

I CAN DELIVER ALL KINDS OF SOFTWARE IN NO TIME—IF IT DOESN'T HAVE TO WORK.

Similarly, some commercial project-estimating packages let managers explore trade-offs among these parameters when they establish the initial project estimates and plans, but they rarely allow for dynamic renegotiations once the project has commenced. Renegotiation may not be all that important on a project that only takes three months. But if a project lasts more than a year or two, renegotiation is almost inevitable in today's turbulent business environment.

Although the precise nature of the mathematical relationships has yet to be developed in detail, we have enough information today — especially from the work of such metrics experts as Larry Putnam, Howard Rubin, and Capers Jones — to provide a reasonable basis for a rational discussion of the issues with our customers. The biggest difficulty, I believe, is one of politics and management. Getting our customers to engage in a rational negotiation will probably require some extensive education, but getting our project managers to negotiate in this fashion will be equally difficult.

It is indeed difficult to say to a customer, "I'm going to deliver a system to you in six months that will have 5,000 bugs in it, and you're going to be *very* happy!" But that may well be the world many of us live in for the next several years. ●

Edward Yourdon is a software-engineering consultant and author who developed the Yourdon method of structured systems analysis and designed and codeveloped the Coad/Yourdon method of object-oriented analysis and design. He edits three software journals: American Programmer, Guerrilla Programmer, *and* Application Development Strategies. *He can be reached at 71250.2322@compuserve.com.*