

## Searching for the Holy Grail of Software Engineering

‘But my project is different’ really is a valid response.

*“A strong method, like a specific size of wrench, is designed to fit and do an optimal job on one kind of problem; a weak method, like a monkey wrench, is designed to adjust to a multiplicity of problems, but solve none of them optimally.”*

—Vessey and Glass, 1998

For decades now, computer science academics have been looking for the Holy Grail of software engineering—the one true approach to building software systems that can be applied, universally, to any and all software projects.

And for just as many decades, software practitioners have looked at these proposed Holy Grails and muttered something that sounds very much like “...but our project is different.”

Those computer science academics, of course, are not terribly impressed by that response. They sort of “nudge-nudge, wink-wink” at each other, as if to say “there go those ignorant practitioners again, resisting what is good for them.” And, as often as not, they mutter something about the practitioner use of the term “ad hoc solutions.” When you hear “ad hoc,” you know the computer science acade-

mic who said it has issued the ultimate insult. “Ad hoc,” in these academic circles, is the ultimate dirty word, the final “j’accuse” of derision.

Over the years some things have begun to change in this regard. Now, there are more computer science academics who are willing to consider the possibility that different problems require different solution approaches.

There has been, for example, a positive response from many readers of [4], which takes the position that problem-focused solutions are strong, and problem-independent solutions are weak, when viewed from the historic perspective of the problem-solving discipline (this column’s introductory quote is taken from that article). But still, as often as not, you will find the same academics who are willing to entertain the possibility that different kinds of problems require different solution approaches, returning like salmon to their spawning area to the notion of so-called “meta-approaches;” collections of those

problem-specific approaches with a problem-independent facade. In the discipline of method engineering, for example, which has come around to the point of view that methodologies must be tailored to the problem domain at hand, there is almost always—in the literature on the subject—someone who is proposing a “meta” approach to those tailored methodologies, as if there is still a generic way out if we only apply sufficient brilliance.

The term “ad hoc” itself is an interesting case in point. As noted, many computer scientists use it as a term of derision, interpreting the term to mean “unstructured” and even “chaotic.” But the dictionary has an entirely different definition for the term. In all the dictionaries I referred to—and I have looked in many because the dictionary definition I found is so different from common computer science usage—the term means “focused on the problem at hand.” Oh, really? That term of derision has thereby become an acceptable approach to problem solving, one our opening quote would even call “strong.”

All of these thoughts simmer beneath the surface of my many

# Practical Programmer

**When you hear the term “ad hoc,” you know the computer science academic who said it has issued the ultimate insult. “Ad hoc,” in these academic circles, is the ultimate dirty word, the final “j’accuse” of derision.**

beliefs about the computing field. They account for my belief in software practice, for example, in the face of tremendous accusations from some theorists who call it a field in crisis. They account for my reversal of position on the field of method engineering; I have always been dubious about what Tom DeMarco refers to as “Capital-M” methodologies, on the grounds its adherents insist users of the methodologies adhere to both the letter and the spirit of their prescriptions. But now that I understand method engineering supports the notion of tailored methodologies—the kind DeMarco refers to with a small “m”—I can believe the methodologies are beginning to offer some real-world, useful concepts practitioners may be able to put to good and (especially) creative use.

As I said, these thoughts often simmer beneath my surface. But every once in a while, they bubble to the top. Here’s the latest example, an occurrence that caused me to renew my rant on the subject of problem-focused solution approaches, the trigger that caused me to write this column.

The *Wall Street Journal* (*WSJ*) engages in some of my favorite journalism. Its editorial pages may reflect opinions somewhat to the right of Genghis Khan on occasion, but its news columns are full

of wonderful, insightful, human-interest stories about many of the most critical issues of our day.

Recently, in the face of booming interest in all the topics e-commerce has spawned, the *WSJ* published a special section on the subject. And, among the many topics covered in that section was an analysis of the high failure rate of so many of those e-business startups [1]. There were plenty of reasons given for the failures, the ones you have probably read about in other literature: poor business planning; focusing on vision more than profit; management by techie; failure to acknowledge the power of the status quo; undercapitalization; overspending; unrealistic expectations. These sort of things. (For a more thorough list, see [3].)

But at the end of that laundry list was another reason, one that resonated with my own (very biased) beliefs: failure to understand the application domain of the underlying problem. Toy e-tailers who don’t understand the low margins and advance purchasing requirements of the toys business [2]. Healthcare e-sites that fail to understand the unique requirements of the medical profession (and believe me, there are plenty of them). B2B businesses that assume their potential customers are ready for an electronic hook-up-buyers-and-sellers approach, without

understanding that “hook-up” means different things in different businesses. One company observed its staff “has to be expert in linking buyers and sellers in hundreds of categories,” and ties its success to “its track record as a brick-and-mortar firm” (it has built automated tools, for example, that capture its traditional strength in “helping buyers buy the right motor from the right specifications”).

There it is, once again. If you know how to provide solutions in the computing field, you are only halfway to becoming a computing expert. You also have to know how to address problems. And you need to understand—let me emphasize the point—that *solution approaches must take into account the nature of the problem at hand*. Even if some would derisively call this way of conducting business “ad hoc.” **C**

## REFERENCES

1. Anders, J. B2B: Yesterday’s darling. *Wall Street Journal* (Oct. 23, 2000).
2. Bannon, L. Toys: Rough play. *Wall Street Journal* (Oct. 23, 2000).
3. Glass, R.L., Ed. *ComputingFailure.com*. Prentice-Hall, Englewood Cliffs, NJ, 2001.
4. Vessey, I. and Glass, R.L. Strong vs. weak approaches to systems development. *Commun. ACM* 41, 4 (Apr. 1998), 99–102.

**ROBERT L. GLASS** (rlglass@acm.org) is the publisher of the *Software Practitioner* newsletter, and editor emeritus of Elsevier’s *Journal of Systems and Software*.

© 2002 ACM 0002-0782/02/0500 \$5.00