Robert L. Glass

The Mystery of Formal Methods Disuse

A story of zealotry and chicanery.

"Formal methods can ... be ***infiltrated*** *into software development activities…"*
"I have suggested ways of ***intruding*** *the use of formal notations…"*
—[1] (EMPHASIS ADDED)

Formal methods have been around for a long time. My recollection of the first article on formal approaches, something on proof of correctness, is one that appeared in the computing literature in the late 1960s. Formal methods have not, during that extended period of time (well over 30 years by now), had any significant impact on the practice of software engineering. It is difficult to imagine anyone who is not curious as to the reason why that is so.

Advocates of the formal approaches—and there are plenty of them—say or imply it is the ignorance and intransigence of practitioners that explains their lack of use of formal approaches. Opponents of the formal approaches—and opposition is considerably more subtle than the advocacy we see—say or imply it is the impracticality of the approaches themselves that prevents their use. In this column, I discuss the mystery of formal methods disuse through the lens of a particular article supporting that use. The referential article, by John Wordsworth, although perhaps extreme in its advocacy, is representative of the sometimes-shrill verbiage of those who cannot imagine why formal methods remain unused on the practitioner's shelf [1].

The Wordsworth article acknowledges the lack of formal methods acceptance in practice, saying things like "there is not much use of formal methods for developing substantial amounts of software." It also does something particularly vital, providing a definition of what it means by formal methods ("a formal method of software development is a process for developing software that exploits the power of mathematical notation and mathematical proofs"). It goes on to state that formal specification and formal verification are the two primary examples of these formal methods. (The reason this definition is vital is that some writers on formal methods expand the subject far beyond any manageable meaning—one writer said "any rigorous approach" could be considered a formal method, for example).

The two key sections of the article are the one on "benefits of formal methods," and the one on "inhibitors to the use of formal methods." These two sections, in fact, contain the reasons I read the article initially, and which motivated me to save the article for several years.

JASON SCHNEIDER

What is really needed is not a rigorous/rigid specification, but one that encompasses the problem evolution that inevitably occurs.

What are those benefits? Almost all of the results cited are from an IBM CICS project that used formal specification (but not formal verification) techniques, and found that the specifications were considerably more understandable than more informal specifications would have been; analyst confidence was higher that the specifications identified the key elements of the problem to be solved; fewer errors resulted in the final product; and ongoing maintenance of the product cost half what it otherwise would have.

Those benefits are certainly significant. But there are several troubling aspects about these findings. Fundamentally, almost no advocates of the formal approaches have been able to cite any other projects of significance where comparable, objective results about the value of formal methods were obtained. (I am a student of evaluative research, and I have been looking for them—without success.) In addition, the CICS project citations that I have seen have not been published in the mainstream computing literature (those cited in [1] include an internal IBM report and a conference paper); the one project citation with which I am familiar appeared, not in the academic literature, but in a practitioner journal (*The American Programmer*). And finally, the findings of this particular study have been questioned in the software engineering literature on the grounds of a weak research approach.

Consider the benefits the author presented: only one study is cited, one whose research approach has been questioned. Nevertheless, the benefits, as stated in this article, appear to be well worth achieving.

Moving on to the section on inhibitors of use, the opening salvo of this section is that "some of the apparent inhibitors are merely excuses…," an indication of the objectivity of the author in presenting them. The article goes on to list these inhibitors: "software engineers are not routinely trained to make appropriate use of formal methods"; they are "not taken seriously by … teachers or … practitioners…"; and "customers do not like to be tied down to function, they prefer to keep things vague…"

I will address each of those inhibitors in turn.

Regarding the first two, which are about "training" and "teaching," I disagree with the author. Look at the educational system by means of which we produce software engineers. Most data we have suggests that, although software folk emanate from a variety of academic educational programs, the majority graduates in computer science. But, according to my observations, computer scientists have been teaching those same formal approaches for most of the years since that first article on the subject was published. Certainly, there has been a solid 20+ years of formal methods education worldwide (European computer scientists, for example, are even more enthusiastic about those formal approaches than their U.S. counterparts.) That means that, far from being ignorant of formal approaches, most contemporary software practitioners have been exposed to and educated in them.

Regarding those customers who prefer to keep things vague, that is a somewhat absurd claim. It implies, of course, that customers could do better in specifying requirements, but prefer not to. What in fact most practitioners tell me about specifications is that the needs of the customer evolve over time, as the customer comes to learn more about solution possibilities, and that what is really needed is not a rigorous/rigid specification, but one that encompasses the problem evolution that inevitably occurs.

Reviewing these inhibitors, then, I would simply disagree with those that Wordsworth identified. There is little reason to assume that education and educators have failed the formal methods movement, and there is of course no reason at all to assume that customers prefer vagueness.

Wordsworth's article, in its final substantive section, comes to grips with its goals. This section is called "a plan for infiltration." Here the author suggests force-fitting formal methods to practice. He advocates something he calls a "Personal Formal Methods Process," basing his description on the popular Personal Software Process work of Watts Humphrey (see www.sei.cmu.edu/tsp/watts-bio.html). I will not dwell on what the author proposes therein, except to note a claim that because "programmers are often given incomplete requirements," the solution is to "exert tactful pressure on those providing the requirements to be more precise about them." I focus, instead, on the use of two particularly interesting words the author uses in this material.

Note that in the two quotations at the beginning of this column, the key boldface italicized words are "infiltrating" and "intruding." Those words struck me as curious choices when I first read the article. To see why they seemed curious, I turned to my favorite dictionary to supplement my first impression. "Infiltrate," it says, "is to penetrate surreptitiously into enemy-held territory." "Intrude," it says, is "to put or force in inappropriately, especially without invitation, fitness, or permission…".

Oh, now I understand. Formal methods advocates, who have failed for decades to make an impact on practice through the front door, are now turning to "surreptitious" and "inappropriate" "force" (in spite of the lack of "fitness" of their concepts).

Now I think you see why I held onto this article for such a long time, contemplating what to do with it. In the end, the result of that contemplation is this column. And the message I express here is: "Practitioners, beware of formal methods zealots. If they can't make their case on the merits, they may well resort to chicanery." ■

References

1. Wordsworth, J.B. Getting the best from formal methods. *Information and Software Technology 41*, 14 (Nov. 1999), 1027–1032.

Robert L. Glass (rlglass@acm.org) is editor emeritus of Elsevier's *Journal of Systems and Software*, publisher/editor of *The Software Practitioner*, and a longtime member of, and believer in, software's practice.

Corrections

The July article "Improving the Quality of Business Object Models Using Collaboration Patterns" (page 81) inadvertently repeated the opening paragraph of the preceding article. The "Security Watch" column (page 25) in that issue incorrectly identifies the acronym HIPAA; the correct definition is Health Insurance Portability and Accountability Act.

Communications editors regret these errors. Corrected versions of this material appear in the ACM Digital Library.