The Relationship Between Theory and Practice in Software Engineering

Robert L. Glass

Reality is the murder of a beautiful theory by a gang of ugly facts.

I want to say something radical at the outset of this new incarnation of an old *Communications* column. I believe in software practice, and I believe in software practitioners. And, as a corollary to those beliefs, I do not agree with those who put down software practice and software practitioners.

There was a time, a half-dozen years ago, when a software engineer taking that position would have been laughed out of the halls of academe and the pages of the computing literature. Cries of "software crisis" opened nearly every article on any software engineering topic, put there by an author who believed his (or, occasionally, her) new proposal was the solution to this apparently rampant crisis. Software was always over budget, behind schedule, and of low quality, the crisis thinking said. How could anyone possibly believe in practice or practitioners?

Fortunately, these things are changing. The computing world—and especially the societal worlds at ACM and IEEE—are beginning to be open to what was once only a contrarian viewpoint, that software practice is doing just fine, thank you. A few respected voices are beginning to cry out that software practice is *not* in crisis, and in fact is doing a pretty good job. Such application domain areas as aerospace, banking, process control, productivity tools, and reservation systems are thriving, woven successfully into the threads of our daily lives. And computing folks such as Al Davis, Peter DeGrace, Tom DeMarco, and Nicholas Zvegintzov are not bashful about pointing at software's successes, and decrying those who cry "crisis."

Russ Willms

Some, in fact, are willing to go even further. Not only is practice OK, these people say, but there are some important flaws in the theory and research area of the field that may be of more concern than the software crisis ever really was. Theorists who fail to evaluate their ideas in a practical setting before advocating them are of particular concern. Fred Brooks, Norm Fenton, Dave Parnas, and yours truly have taken this viewpoint in print and public forums a few times, and been surprised at the force of both those who support, and those who oppose, this viewpoint.

So this column will, in the future, be a sort of ode to software practice. I want to talk about some of the good things that practitioners do, about

some of the better bridges between an aware world of theory and a receptive world of practice, about some interesting software practitioners, about some best-of-practice findings, and about lessons practitioners have learned that will benefit both other practitioners and theoreticians willing to listen to them. And I hope you'll want to help me in this mission. I solicit your ideas, your columns, and—yes—even your criticism. After all, this new field of software is roughly only 40 years old, and there is far more speculation than truth in the things both theory and practice have come to "know" thus far.

To set the stage for all of this, in this first column I want to talk a little bit about the relationship between theory and practice in the field of software engineering. This relationship is important because some appropriate analysis will tell us that in this new field of ours, sometimes it is preferable to listen to theory for new ideas, and sometimes it is preferable to listen to practice, and the aware computer person should know when each is appropriate. This, of course, applies to both theoreticians and to practitioners.

Perhaps you are surprised anyone would ever think that practice could lead theory? You shouldn't be. In any new discipline, it is often true people do things for which theory has no explanation and provides no foundation, and theory evolves only after practice has demonstrated that something works. It happened in aerodynamics, where the invention of the airfoil preceded and motivated the science of aerodynamics. It happened in thermodynamics, where the invention of the steam engine preceded and motivated the science of thermodynamics. It happened in the field of computing only twenty-something years ago, when the time-sharing system was developed and marketed before there was a solid theoretical background for it. And it was an essential part of the early history of the field, when computers were being sold in the marketplace in the 1950s, and the academic discipline of computer science did not appear on the scene until a decade later.

Of course, the history of most fields is more complicated. Theory and practice really tend to evolve hand-in-hand. Early computing work was done in academic laboratories, far before there was much in the way of practical application of the results. This in turn spawned practical application, which in turn spawned academic programs. Theory and practice do and must progress together; it is foolish to believe practice always leads theory, but it is also foolish to believe the opposite.

Past is, of course, prologue. But how important is this philosophic history to the computing field circa 1996? I think its importance is both profound and underestimated.

The importance is profound because there are times when theory has answers to questions practice has not yet asked, but there are also times when practice has answers to questions theory has not yet raised, and it is important to know who to listen to and when to listen. From which source should we expect the wisdom of the next millennium to emerge? The notion of "best-of-practice" concepts emerges from the belief that practice *can* lead theory.

The importance is underestimated because most of us have an intuition that says theory in general is ahead of practice, and it is hard to let go of that intuition even in the face of evidence to the contrary. It may be sensible to believe there are times when we should listen to practice first, but it is hard to drop a deep-seated, almost religious belief that theory is where all the answers come from.

NOT ONLY IS PRACTICE OK, BUT THERE ARE SOME IMPORTANT flaws in the theory and research area of the field that may be of more concern than the software crisis ever really was.

So let me get specific. The following are some topic areas from the field of software engineering where I believe practice leads theory. In these areas, I believe, both theorists and practitioners should start with the premise that the knowledge of practice is superior and more advanced than the knowledge of theory. I don't bother to identify the other areas, where theory leads practice, because 1) there are probably more of them, and 2) it is important to stimulate the somewhat-contrarian thinking that practice *can* lead theory.

Here is my list of candidate topics where practice leads theory in software engineering:

1. *Software design.* Practitioners have designed software since the first user brought the first problem to a software specialist over 40 years ago. But theory is still wrestling with what design really is. The theory of design tends to focus on methodologies and representations, because those are "hard" and therefore teachable/testable topics. But the best methodologies and representations in the world will not necessarily lead us to superior designs. There is something else to the act of design, and practitioners have found it, while theorists are still searching. In fact, in many academic institutions the best answer to the question "How do we teach design?" is "In a laboratory setting using mentoring." In other words, let those who can practice design teach those who are learning how.

2. *Software maintenance.* Practitioners have been maintaining software for as long as they have been designing it. For years, theory had little interest in maintenance, believing it was either an uninteresting subset of development, or a nonintellectually challenging task. That has only changed in the past few years; it is still true there is a rich lore of maintenance knowledge in practice, and only a slowly emerging lore of knowledge from research.

3. *User interface.* This is a topic area where theory and practice have progressed together. The pioneering work at Xerox PARC has led to the friendly interfaces of the 1990s, commonly available now on all commercially available computers. Continuing exploration is occurring in both industry (e.g., Microsoft's "Bob") and in academe.

4. *Programming in the large.* The problems of today are up to 50 times larger than the problems of yesteryear, and the complexity of software solutions is growing at an exponentially faster rate than the growth of problems. Many solution approaches simply don't scale up to these massive problems, and one of the most important issues in software engineering is to define what will and what will not scale. Whereas practice is involved with these huge problems on an ongoing basis, because of its expense theory is seldom able to do the required research in the large. Thus, now, and in the foreseeable future, practice leads theory in the understanding of programming in the large.

5. *Modeling and simulation.* In order to understand complex problems, practitioners often model the problem domain and provide a simulation of solution approaches in order to determine solution feasibility. Since theory is rarely involved with large and complex projects, it is less familiar with the role that modeling and simulation can play.

6. *Metrics.* Theory and practice are traveling very different roads on the topic of software metrics. Theoreticians have proposed a number of metrics, but most of them are unverified and there are considerable differences among theorists as to their value. Practitioners, on the other hand, seldom use metrics, but when they do the set used is often unrelated to the set proposed by theorists. At the present time, it is difficult to say whether theory or practice is leading in the topic of metrics, but it is possible to say that the field is still in turmoil. In the next decade, we should find out which is ahead, theory or practice.

As I've mentioned, these are candidate topics. You may or may not agree with what I have selected; you may or may not have others to propose. But what is important about this list, or your list, or anyone else's list, is that it defines whose expertise is best in the topic areas in question. If I want to find out about programming in the large or software maintenance or whatever topic is on this list in the 1990s, there is no question in my mind that I will seek my answers in practice, not in theory.

As I said at the outset of this column, this is still somewhat contrarian thinking. Some practitioners and many theoreticians will probably not agree with or like what I have said. I would be interested in what you think. ◻

Robert Glass *is president of Computing Trends and publisher and editor of the* Software Practitioner. *He welcomes feedback: 1416 Sare Rd., Bloomington, IN 47401.*