Editor in Chief: Steve McConnell ■ Construx Software ■ software@construx.com

Closing the Gap

Steve McConnell

Years ago, Fred Brooks commented, "The gap between the best software engineering practice and the average practice is very wide—perhaps wider than in any other engineering discipline." The past few years have seen a proliferation of books on project management, requirements, architecture, design, testing—nearly every area of software engineering. But within the companies I visit in my consulting business, I rarely see software engineering best practices being used. Increasingly, I ask myself, "Why aren't people using the numerous good software engineering practices that are now so readily available?"

Classic barriers to innovation

A conventional answer to this question is that many of these practices simply aren't yet mature. When presented with a new practice, software practitioners tend to ask tough questions such as these:[1]

- Do experimental results prove conclusively that the practice will work in the field?
- Are successes a result of the practice itself, or might they be the result of the people using it?
- Is the practice complete, or does it need to be adapted or extended before it can be applied?
- Does the practice have significant overhead (training, documentation) that offsets its value in the long run?
- If the practice was developed in a research setting, does it apply to real-world problems?
- Does the practice generally slow down the programmers?
- Can the practice be misapplied?
- Is information available about the risks involved with using the practice?
- Does the practice include information about how to integrate it with existing practices?
- Must the practice be applied in its entirety to realize significant benefits?

These are all fair questions, and I think it's healthy for practitioners to ask them. Indeed, part of *IEEE Software*'s mission is to help our readers answer these questions. However, the practices I'm thinking of are hardly new, and, for many of them, I believe many of these questions have already been answered. Table 1 lists numerous practices that leading organizations have understood well and deployed for decades.

In the management arena, we've had automated estimation tools since the early 1970s, but most projects don't use them. Measurement has been a key topic for 25 years, but few organizations collect quantitative data on their projects. I still see software developers housed in open work bays or cubicles far more often than I see them working in private or semiprivate offices—even though research about the effect of physical environment on productivity has been conclusive for more than 15 years.

One of the most fundamental practices in software engineering is change control, especially as it relates to software requirements. I teach a two-day workshop based on my book *Software Project Survival Guide* (Microsoft

Table 1

Year of introduction of rarely used software best practices

Best practice	Year first described in print or first available commercially
Project planning and management practices	
Automated estimation tools	1973
Evolutionary delivery	1988
Measurement	1977
Productivity environments	1984
Risk management planning	1981
Requirements engineering practices	
Change board	1978
Throwaway user interface prototyping	1975
Joint Application Design	1985
Design practices	
Information hiding	1972
Design for change	1979
Construction practices	
Source code control	1980
Incremental integration	1979
Quality assurance practices	
Branch-coverage testing	1979
Inspections	1976
Process improvement	
Software Engineering Institute's Software Capability Maturity Model	1987
Software Engineering Process Groups	1989

Press, 1998). When I originally developed the workshop, I included a module on change control, because I could easily pull together the necessary materials and I was working under some deadline pressure. I assumed that it would be too basic for most of my students and that I would need to replace that module as soon as I had time. To my surprise, three years later, after teaching the class about 20 times, I've had only one group of students in which more than half were already using change control. Change control has been described in the software engineering literature since 1978, but the basic practice has been employed in other branches of engineering for at least 50 years. All the tough questions listed earlier were answered for change control decades ago. Considering the practice's central role in software project control, I am puzzled about why software projects don't use this fundamental practice universally.

Barriers to software innovations

Software presents unique challenges to adopting better practices. One challenge is a lack of awareness that good practices exist. Where, ideally, should someone learn about fundamental software engineering practices? In most fields, we expect universities to provide education in the fundamentals. Until very recently, however, most undergraduate degree programs related to computer programming have not including training in these basic practices. Additional university programs are coming online

IEEE Software

DEPARTMENT EDITORS

Bookshelf: Warren Keuffel, wkeuffel@computer.org

Construction: Andy Hunt and Dave Thomas, Pragmatic Programmers, {Andy, Dave}@pragmaticprogrammer.com

Country Report: Deependra Moitra, Lucent Technologies d.moitra@computer.org

Design: Martin Fowler, ThoughtWorks, fowler@acm.org

Loyal Opposition: Robert Glass, Computing Trends, rglass@indiana.edu

Manager: Don Reifer, Reifer Consultants, dreifer@sprintmail.com

Quality Time: Jeffrey Voas, Cigital, voas@cigital.com

STAFF

Senior Lead Editor
Dale C. Strok
dstrok@computer.org

Group Managing Editor
Crystal Chweh

Associate Editors
Jenny Ferrero and Dennis Taylor

Staff Editors
Shani Murray, Scott L. Andresen, and Kathy Clark-Fisher

Magazine Assistants
Dawn Craig
software@computer.org

Pauline Hosillos

Art Director
Toni Van Buskirk

Cover Illustration
Dirk Hagner

Technical Illustrator
Alex Torres

Production Artist
Carmen Flores-Garvey

Executive Director
David Hennage

Publisher
Angela Burgess

Assistant Publisher
Dick Price

Membership/Circulation Marketing Manager
Georgann Carter

Advertising Assistant
Debbie Sims

CONTRIBUTING EDITORS

Greg Goth, Denise Hurst, Gil Shif, Keri Schreiner, and Margaret Weatherford

Editorial: All submissions are subject to editing for clarity, style, and space. Unless otherwise stated, bylined articles and departments, as well as product and service descriptions, reflect the author's or firm's opinion. Inclusion in *IEEE Software* does not necessarily constitute endorsement by the IEEE or the IEEE Computer Society.

To Submit: Send 2 electronic versions (1 word-processed and 1 postscript or PDF) of articles to Magazine Assistant, *IEEE Software*, 10662 Los Vaqueros Circle, PO Box 3014, Los Alamitos, CA 90720-1314; software@computer.org. Articles must be original and not exceed 5,400 words including figures and tables, which count for 200 words each.

each year, and I think the lack of infrastructure is due simply to software engineering's being such a young field.

In the absence of university education systems, we might expect software-producing companies themselves to provide supplemental training. In fact, a few leading companies do train their software engineers, but not to an extent great enough to ameliorate industry-wide software problems.

In less advanced companies, the lack of training has been more difficult to address. Before a manager can prescribe training, he needs to know that a field of knowledge is deep enough to need training. Managers who came up through the technical ranks 20 years ago, or even 10 years ago, might underestimate the depth of knowledge in modern software engineering. Many software managers are not themselves well trained enough to realize that their staff needs training.

Calling all experts

These are all descriptions of what has not been done, but they still leave open a basic question: Why don't software engineers—who are some of the brighter people on the planet—seek out better methods of doing their work? We're all aware of the pain arising from not using these practices. So why don't practitioners more actively seek them out and use them?

With all the advances during the past several years, it appears that the challenge for the software industry has shifted from good-practice development to good-practice deployment.

What do you see as the barriers to deployment of good practices? How do you think good practices can be deployed more quickly? I invite your comments.

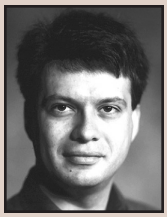
Reference

1. S.A. Raghavan and D.R. Chand, "Diffusing Software-Engineering Methods," *IEEE Software*, vol. 6, no. 4, July 1989, pp. 81–90.

Andy Hunt and Dave Thomas Join *IEEE Software* Editorial Board

Andy Hunt and Dave Thomas, founders of The Pragmatic Programmers LLC, recently joined *IEEE Software*'s Editorial Board.

Prior to joining Pragmatic Programmers, Hunt worked in various senior positions at Discreet Logic, Alias Research, Philips Medical Systems, and AT&T. He received his BS in information and computer science at the Georgia Institute of Technology. He is a member of the IEEE Computer Society, the ACM, and Independent Computer Consultants Association.



Andy Hunt

Thomas cofounded and ran a software company in the United Kingdom prior to joining Pragmatic Programmers. Thomas holds an honor degree in computer science from London University. He is a member of the IEEE Computer Society and the ACM.

Dave Thomas

Hunt and Thomas have coauthored two books, *The Pragmatic Programmer: From Journeyman to Master* (Addison-Wesley, 2000), and *Programming Ruby: The Pragmatic Programmer's Guide* (Addison-Wesley, 2001). They have also written a number of articles together, including "Learning to Love Unit Testing" for *Software Testing and Quality Engineering Magazine* (Jan. 2002) and "Programming in Ruby" for *Dr. Dobb's Journal* (Jan. 2001). Individually and together, they have also given numerous talks and tutorials at conferences and workshops.

Contact Andy Hunt at andy@pragmaticprogrammer.com and Dave Thomas at dave@pragmaticprogrammer.com; www.pragmaticprogrammer.com.