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

I Know What I Know

Steve McConnell

In most of my columns, I write about topics I know about. I am becoming increasingly aware of how many topics I don't know. This column describes some questions that I personally would most like to have answered.

How important is software construction?

Software construction has been the awkward stepchild of software engineering for decades. This puzzles me because software construction is the only activity that's guaranteed to be done on every project. Code is the center of the software universe. Process advocates tend to downplay coding, focusing more on the documentation that occurs before and after coding than the code itself. Academics have focused for decades on eliminating coding. My efforts to recruit construction articles for *IEEE Software* have been an uphill battle because it seems that very few people want to write about coding.

Because construction happens on every project, why do some experts downplay its importance? I suspect that a lot of the energy behind two recent movements—open source and Extreme Programming—arises because each recognizes the central role that coding plays in software development. This is a breath of fresh air to software developers who never forget that software is code.

How do you manage multiple releases?

Managing complicated versions is one of the most significant challenges in software I see today. Released versions can vary by features, hardware platform, language, culture, customer, and many other factors. I commonly see companies actively supporting and enhancing more versions of their system than they have programmers. What are the most useful strategies for managing large numbers of concurrent releases based on the same code base? What are the strategies for managing requirements, design, and tests that go along with those releases?

Why doesn't everyone use revision control software?

A more mundane question I have about configuration management is, Why are some people still not using revision control software? Ten years ago I debated whether I should write about revision control software in my book *Code Complete*. I assumed revision control software was so common that discussing it would be passé. After talking with literally hundreds of developers and managers the past several years, however, I am convinced that somewhere between 10 and 50 percent of software organizations still do not use revision control software. Why?

How is popular software designed?

I would like to see the designs for some of the world's best known software. What does the design for the Sabre airline reservations system look like? What about the US air traffic control software? What are the guiding principles of the design of Windows 2000, PC-DOS, and IBM OS/360? What about the space shuttle flight control software? What are Excel's major design challenges? What about SAP? What about Amazon.com's Web site?

If these popular software products were buildings, I would be able to see at least some of their designs. For example, the National Archives and Records Administration Web site (www.nara.gov) contains plans for

IEEE Software

28,000 buildings going back 150 years. Architectural techniques are not treated as "proprietary" because the public has a stake in building safety. Software has become so pervasive in modern life that I think the public has a similar interest in the design of some of the most pervasive software systems. What would it take to see these designs?

How big are popular programs?

How big are today's common systems, and what did it take to build them? I'd like to see cost, effort, schedule, lines of code, and defect counts for major applications such as Windows 2000, TurboTax, Norton Anti-Virus, Adobe PhotoShop, Excel, and SAP. Having a list of the management parameters of well-known applications would help companies create meaningful ballpark estimates for their own systems. It wouldn't replace detailed estimation, but would help reduce the common factor-of-2 to factor-of-10 errors in initial ballpark estimates.

Why do software professionals still fall for silver bullets?

Fifteen years ago in his classic article "No Silver Bullets,"[1] Fred Brooks predicted that no single tool or practice would produce an order-of-magnitude improvement in quality or productivity over a 10-year period. In spite of repeated "silver bullet" claims for innovations ranging from automatic programming to component-based development to object-oriented programming to CASE tools (the list is nearly endless), no single tool or practice has risen to the level Brooks described. Time has proved Brooks' prediction correct.

Software professionals have been burned time and time again by exaggerated claims for new tools and practices. My question is, Why are so many smart, experienced software professionals still so gullible about silver bullets?

Do compensation structures in software organizations make any sense?

Many companies continue to treat managers as hierarchically superior to technical staff and pay them more. The idea that companies turn good programmers into bad managers is so common it is a cliché. Software managers have a broad span of control, whereas technical staff tend to be much more narrowly and deeply focused, and I think this might be part of the reason why managers are paid more. But professional athletic teams value depth of skill at least as much as breadth. Star players earn more than star managers. Why don't software companies do the same?

Similarly, the ten-fold difference in productivity between best and worst software developers has been well documented. Yet I routinely talk to managers who say their companies will not pay above market to attract top developers. Considering that the difference in compensation between best and worst developers is only about 25 to 50 percent, while the difference in performance is about 1,000 percent, why don't all companies make at least some attempt to hire from the top of the talent pool?

How good are most software companies?

In the software process area, the Software Engineering Institute reports that organizational process maturity has increased dramatically. In 1991, approximately 80 percent of organizations assessed were at Capability Maturity Model Level 1.[2] By 2001, only 38 percent of organizations assessed were at CMM Level 1. My question is, How many companies are really at CMM Level 1? Only a small number of companies report assessment data to the SEI (less than 500 companies in 15 years). What are the CMM levels of the thousands of companies that don't report their results to the SEI? What about the companies that haven't heard of the CMM? (Actually, I think I can guess the answer to this question!)

What are software's real best practices?

Although software people sometimes have a tendency to latch onto "one size fits all" solutions, I think most people realize that different application domains call for different software development approaches. It's natural and appropriate to use one set of tools and practices to develop avionics software and a different set to develop video games.

Certain clusters of practices seem to work well within particular domains. For embedded systems, phased, gated processes with lots of up-front requirements work and design seem to work well. For software products companies, code-focused development efforts that use highly committed individuals, have a close working relationship with marketing, and perform extensive independent testing are appropriate. For in-house business systems, executive sponsorship, steady end-user involvement, moderate levels of requirements documentation, and developer testing seem to be key. These broad strokes are recognizable to people working in these areas.

My question here is, Can we map out the applicability of these practices in detail? Are there some clusters of practices that interact synergistically? If some practices do interact synergistically, are they always synergistic, or only when used to develop certain kinds of software? Are there practices that are best practices in some contexts and worst practices in other contexts? Does *any* single practice constitute a best practice in all areas?

What do you know?

I know what I know, and now I'd like to hear what you know! If you have thoughts about these questions, I'd love to hear from you at steve.mcconnell@construx.com.

References

1. Frederick P. Brooks, "No Silver Bullets—Essence and Accidents of Software Engineering," *Computer*, vol. 20, no. 4, Apr. 1987, pp. 10–19.
2. "Process Maturity Profile of the Software Community 2001 Year End Update," Software Engineering Measurement and Analysis Team, Software Engineering Inst., pdf/2002mar.pdf; www.sei.cmu.edu/sema/pdf/2002mar.pdf.

Upcoming Topics

July/August: Initiating Software Product Lines

Adopting a product line perspective on software development affects the "usual" way of doing business at the organizational and technical management levels as well as at the technical software engineering level. Look for techniques, case studies, and more.

Out of the Box: Solving problems in our world of fast-changing technology and expanding information requires creative and innovative ideas. We will explore some of the qualities and practices that can help us break through traditional modes of thinking.

Sept./Oct.: Educating Software Professionals

What do software engineering professionals need to know, according to those who hire and manage them? This issue will focus on the methods and techniques for enhancing software education programs worldwide—academic, re-education, and alternative.

Nov./Dec.: The Business of Software Engineering

Software professionals can benefit both themselves and their companies by weaving business and economic considerations into their software engineering decisions. This issue will explore what it takes to support business success.