We can certainly debate how often software development is treated as engineering, but any debate about whether software development can be treated as engineering is a few years out of date. The IEEE Computer Society adopted a Code of Ethics for Software Engineers almost 10 years ago. The Society approved the Software Engineering Body of Knowledge 2.0 in 2004, which was adopted as an ISO/IEC Technical Reference in 2005. Curriculum guidelines and accreditation standards have been set for undergraduate software engineering programs. In the US, the official engineering accreditation board, ABET, has accredited 13 undergraduate software engineering programs since 2003, and in Canada, CEAB has accredited nine such programs. Numerous provinces in Canada license professional software engineers, and professional engineers are chartered in software in England.

At this time, the issue is not debating whether software development can be treated as an engineering discipline; it can be, and it is. The issue is educating practitioners and bringing them up to speed on the current state of the art. This column illustrates that education about the field of software engineering still has a long way to go. Steve McConnell steve.mcconnell@construx.com

The author responds:
The purpose of my article was to discuss the problems that software development is facing, not to discuss whether software development is engineering or not.

The partial list that Steve McConnell suggests consists of activities that help develop better software, but none of them, I opine, really involves “writing or maintaining software.” His partial list goes along with my opinion that there are other engineering disciplines that lay foundations for software development. However, that does not necessarily imply that writing software or, to be more explicit, program coding is engineering.

I am quite disappointed that Mr. McConnell opted to refer to the reality-based experiences and opinions of practicing software developers and other researchers as “myth” and that he went on further to suggest that Computer should not accept opposing views in a debate.

I certainly hope that Computer does not adopt such a policy because it might alienate people who count. For example, I believe that the 2005 ACM Turing Award winner, Peter Naur, opted to resign his ACM membership under a somewhat similar situation (“Computing versus Human Thinking,” Comm. ACM, Jan. 2007, pp. 85-94).

Perhaps it’s appropriate to use Merriam-Webster’s number one Word of the Year for 2006, first introduced by political satirist Stephen Colbert on a Comedy Central broadcast in 2005: truthiness—“truth that comes from the gut, not books.” The truthiness about software development, particularly writing and maintaining software, is that it is engineering. However, the truth for quite a few practicing software developers might be different. Raghavendra Rao Loka rao@acm.org

“Software Development: What Is the Problem?” makes sad reading. The author claims that software development today is not an engineering discipline and implies that it will never develop into one. He is quite correct in the first part of this claim.

There is in fact a way to apply engineering methods to software development: intensive application of the finite-state machine. The FSM, a concept often misunderstood in the software community, provides a theoretically rigorous, clear, and complete way of specifying a complex system’s behavior. But of course such systems must be designed, or modeled, by using many state machines in a well-controlled structure.

Although FSM systems only deal with behavior, to the exclusion of computational algorithms, this is usually the critical area governing software reliability. Such models are always...
platform-independent, and a fixed program executes them as such with no attempt to generate code. Even in today’s high-level languages, code does not scale well to handle complex systems, and it quickly becomes unreadable. A very-high-level language concept is required to handle complexity. Various coded functions are easily added to the “skeleton” the FSM system provides.

As the FSM model is very simple, designers can discuss product specifications (models) with third parties such as the end user during development. In complex projects, it is a pleasure to be able to work at such a high level of abstraction. Projects done in this way normally work correctly when delivered and require little maintenance effort.

Further information can be found in *Modeling Software with Finite State Machines: A Practical Approach* (Auerbach, 2006) and in various published papers. The first part of the book discusses the present situation and endorses the views the author of the *Computer* article expresses so ably. The second part explains in some detail how to apply the FSM concept. The final part of the book describes the StateWorks product to illustrate how the basic ideas, which are not patented, have been applied in practice.

*Ferdinand Wagner and Peter Wolstenholme*  
p.wolstenholme@computer.org

---

**ERRATA**

Due to an editing error, the names of the authors of “Sparse Matrix Computations on Reconfigurable Hardware” (*Computer*, Mar. 2007, pp. 58-64) were inadvertently transposed. The correct listing is:

Gerald R. Morris, US Army Engineer Research and Development Center  
Viktor K. Prasanna, University of Southern California.

In addition, on page 58, the sentence at the bottom of the right-hand column should read: “Huppenthal made this comment at the ARSC reconfigurable computing conference in August 2005.”

Also, please note that on page 61, two symbols in Equation 1 are incorrect. The correct equation is:

\[
X_i^{\delta+1} = \frac{1}{a_i} \left[ b_i - \sum_{j=i; j\neq i}^{j=n} a_{ij}x_j^i \right].
\]

*Computer* regrets these errors.