

## Is Software Engineering as We Know It over the Hill?

Warren Harrison

recently participated in an email discussion with several senior software development experts. The discussion naturally turned to the use of software engineering best practices. One comment in particular struck me (paraphrased here):



*All these ... have strong followings to this day. ... What worries me is "to this day"—just how long in the tooth are we getting here?*

Many of the best practices we follow today were originally proposed decades ago. For instance, Walker Royce first described the famous Waterfall lifecycle model in 1970.<sup>1</sup>

Michael Fagan first described his inspection method—arguably the most effective of all peer review methods—in the open literature in 1976.<sup>2</sup> The insights and techniques that the leaders of the day contributed were welcome revelations. However, how well have these contributions withstood the last three decades? Is classical software engineering as we tend to think of it obsolete? Have these techniques run their course?

### Software's wonder years

To put things in perspective, consider the software industry in the early and mid-1970s when much of what we consider classical software engineering was developed. It differs so much from contemporary software development as to be almost unrecognizable.

Developers generally wrote applications for large, centralized systems, almost always funded by someone with deep pockets such as

the military or large corporations. Systems were usually built to achieve automation—that is, to computerize some task that humans had been doing. So, the developers often could specify the expected behavior in advance (naturally this doesn't mean they always did) by simply observing the manual processes. I don't mean to imply that innovative applications weren't developed during this period nor that automation didn't have its own unique challenges. But most organizations bought into computing in the 1960s and 1970s to make processes previously done by people cheaper and faster.

If you would have asked the average software developer about user interfaces in 1970, he or she would have pointed you to an IBM 029 key-punch machine. Point and click indeed! In fact, the most common software system architecture comprised mainly centralized, batch-oriented automation projects. The system "user" was typically the person who read the reports it generated, seldom even considering issues such as navigation and usability. Programmers were furnished with "output sheets" made up of a grid with 55 lines and 132 columns—the number of lines and rows on a typical sheet of tractor-fed computer paper. This is how an application was "prototyped" for the user.

### Programming environments?

Computer cycles were very expensive. In fact, one of my graduate school professors, John Metzner, suggested in 1980 that the most valuable computing invention that could be made would be a "CPU battery," where programmers could save up all the cycles that were otherwise wasted when the computer was idle. Although time-sharing systems were certainly

### DEPARTMENT EDITORS

**Bookshelf:** Warren Keuffel,  
wkeuffel@computer.org

**Construction:** Andy Hunt and Dave Thomas,  
Pragmatic Programmers,  
(Andy, Dave)@pragmaticprogrammer.com

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

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

**Manager:** Don Reifer, Reifer Consultants,  
d.reifer@ieee.org

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

**Requirements:** Suzanne Robertson  
suzanne@systemsguild.com

### STAFF

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

Group Managing Editor  
**Crystal Shif**

Associate Editors  
**Shani Murray and Dennis Taylor**

Assistant Editor  
**Denise Kano**

Editorial Assistants  
**Rebecca Deuel and Joan Hong**

Magazine Assistant  
**Pauline Hosillos**, software@computer.org

Art Director  
**Toni Van Buskirk**

Cover Illustration  
**Dirk Hagner**

Technical Illustrator  
**Alex Torres**

Production Assistant  
**Monette Velasco**

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

Candace English, 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:** Access the IEEE Computer Society's Web-based system, Manuscript Central, at <http://cs-ieee.manuscriptcentral.com/index.html>. Be sure to select the right manuscript type when submitting. Articles must be original and not exceed 5,400 words including figures and tables, which count for 200 words each.

available by 1970, they were by no means widely accepted as the de facto programming environment. A 1968 study reported (surprise!), "Statistically significant results indicated faster debugging under online conditions ...," yet it also listed several contemporary citations that rejected the use of online programming because it diverted resources from "productive data processing" to "unproductive uses."<sup>3</sup>

These early "programming environments" were not particularly agile, especially if combined with the use of punch cards rather than online text editors. To conserve computer resources, programmers were taught to write their programs out on coding forms prior to entering the code into the editor or having it keypunched. If a simple coding error turned up after the program had been keypunched, inputting the line again and finding which card to replace in the deck were tedious and time-consuming activities. So, programmers usually did *desk checking*—pouring over the coding forms before keypunching to look for syntax and logic errors. Sometimes it would help to have an office mate look over the coding forms as well, a very basic form of peer review.

### Captive consumers

There was no mass market consumer software as we know it today, because there was no consumer mass market. Most users were captives to the application because they needed to use it to perform their jobs, yet they had little say in its acquisition. I well remember my early courses in systems analysis and design, which carefully distinguished "customer" and "user."

Although a fair amount of hobbyist software was produced in the late 1970s, it was, well, hobbyist software. Probably the first true mass market consumer software didn't appear until Software Arts released Visicalc in 1977. IBM didn't introduce the PC until 1982, and it wasn't until 10 years later in 1992 that Microsoft released Windows 3.1 (the first version that allowed multitasking). In 1986, Borland shipped my all-time favorite development environment, Turbo Pascal 3 (compiler, integrated ed-

itor, and debugger), on a single 360-Kbyte (yes, K!) floppy disk. (You can still download Turbo Pascal 3 from Borland's Antique Software page at <http://community.borland.com/article/0,1410,20792,00.html>. The file size is 170,209 bytes.) I used this environment to develop SET Laboratories' first commercial product, PC-Metric for Pascal, which we released (complete with three-ring binder and a 5 1/4-inch floppy disk in a sandwich bag) in 1987.

With no mass market to contend with in the early days, second-guessing the limited user base most applications had was relatively easy.

### Fast forward to today

Software development—both the market and the way we do it—has changed dramatically over the past 35 years. Do the lessons we learned during the 1970s and 1980s still apply?

Some members of our community say no. For instance, some of the principles behind the Agile Manifesto reject the concepts of the classical Waterfall lifecycle model. These principles embrace incremental delivery of working software and encourage changing requirements, an anathema to the Waterfall model. Lightweight informal reviews by programmer pairs replace the use of formal inspections.

The agile viewpoint stems from the myriad dramatic changes we have observed in the industry over the last few decades. Development environments are more agile. Delivery mechanisms nowadays consist of Web-based downloads. Are the days of phased life cycles and formal inspections over?

### Littering and software engineering

A community can have three types of knowledge, according to C.W. Choo:<sup>4</sup>

- *Implicit knowledge* represents individual or group experience and expertise and is difficult to share.
- *Explicit knowledge* is based on formal policies, procedures, instructions, and standards.
- *Cultural knowledge* is the basis for what we think is fair and trustworthy.

We can imagine a process in which the implicit knowledge of a few influential individuals is formalized and becomes explicit knowledge. After a certain period of consistent promulgation, this explicit knowledge becomes cultural knowledge.

A good example of this process involves rural roadside littering in the United States. Up until the 1960s, it was common for drivers to litter—it was simply considered an acceptable practice. I can recall road trips with my parents in the early 1960s in which cigarette butts, candy wrappers, and empty drink cans were nonchalantly tossed out the windows of passing cars.

Some influential people, such as US President Lyndon Johnson's wife, Lady Bird, had implicit knowledge that this was a disagreeable and offensive practice. This led to explicit knowledge, expressed in the form of a 1965 US federal law called the Highway Beautification Act. Within a couple decades, this explicit knowledge evolved into cultural knowledge. Today, most Americans wouldn't think of throwing an empty drink can from their car. It would be hard to imagine how society could be changed to view the notion of littering as acceptable again.

Many classical software engineering practices that were first converted from

implicit knowledge to explicit knowledge in the 1970s have evolved into cultural knowledge. Today, most developers would reject the notion that you can develop a quality software application without progressing through a sequential, phased life cycle. New methods that seem to challenge this culturally accepted practice have transitioned from implicit knowledge to explicit knowledge, but only time will tell if their evolution will continue on to cultural knowledge. Until then, however, I don't think I'd write off classical software engineering techniques. They're cultural.

I'd like to hear from *IEEE Software* readers. Is classical software engineering over the hill? Please write to me at warren.harrison@computer.org. ☺

**References**

1. W.W. Royce, "Managing the Development of Large Software Systems: Concepts and Techniques," *Proc. Wescon*, IEEE CS Press, 1970.
2. M.E. Fagan, "Design and Code Inspections to Reduce Errors in Program Development," *IBM Systems J.*, vol. 15, no. 3, 1976, pp. 182-211.
3. C.H. Sackman, W.J. Erikson, and E.E. Grant, "Exploratory Experimental Studies Comparing Online and Offline Programming Performance," *Comm. ACM*, vol. 11, no. 1, Jan. 1968, pp. 3-11.
4. C.W. Choo, *The Knowing Organization: How Organizations Use Information to Construct Meaning, Create Knowledge, and Make Decisions*, Oxford Univ. Press, 1998.

**Introducing a New Associate Editor in Chief**

When we talk about "knowledge workers," few professions so epitomize the phrase as software development. However, the very phrase suggests that knowledge acquisition is key to the profession. In recognition of this, *IEEE Software* is adding a new associate editor in chief, Donald Bagert, to focus on the topics of software engineering education and training.

Don is a professor of computer science and software engineering at the Rose-Hulman Institute of Technology, where he is also director of software engineering. Don came to Rose-Hulman after 14 years at Texas Tech University. His research interests include software process improvement, software tools for student advising, and software methodologies. Don received his PhD in computer science from Texas A&M University. He is the first licensed Professional Engineer in software engineering in Texas and since Texas is the first state to license software engineers, he is the first in the United States.

Don chairs the steering committee for the IEEE Computer Society Conference on Software Engineering Education and Training (CSEE&T) as well as the Society's CSDP (Certified Software Development Professional) Program Certification Committee ([www.computer.org/certification](http://www.computer.org/certification)). He is a member of the Educational Activities Board and the Professional Practices Committee and a senior member of the IEEE.

EDITOR IN CHIEF:

**Warren Harrison**  
10662 Los Vaqueros Circle  
Los Alamitos, CA 90720-1314  
warren.harrison@computer.org

EDITOR IN CHIEF EMERITUS:

Steve McConnell, Construx Software

**ASSOCIATE EDITORS IN CHIEF**

**Education and Training:** Don Bagert, Rose-Hulman Inst. of Technology; don.bagert@rose-hulman.edu

**Design:** Maarten Boasson, Quaerendo Invenietis  
boasson@quaerendo.com

**Construction:** Terry Bollinger, Mitre Corp.  
terry@mitre.org

**Requirements:** Christof Ebert, Alcatel Telecom  
christof.ebert@alcatel.com

**Management:** Ann Miller, University of Missouri, Rolla  
millera@ece.umar.edu

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

**Experience Reports:** Wolfgang Strigel,  
Software Productivity Center; strigel@spc.ca

**EDITORIAL BOARD**

- Nancy Eickelmann, Motorola Labs
- Richard Fairley, Oregon Graduate Institute
- Martin Fowler, ThoughtWorks
- Robert Glass, Computing Trends
- Jane Hayes, University of Kentucky
- Andy Hunt, Pragmatic Programmers
- Warren Keuffel, independent consultant
- Brian Lawrence, Coyote Valley Software
- Karen Mackey, Cisco Systems
- Deependra Moitra, Infosys Technologies, India
- Don Reifer, Reifer Consultants
- Suzanne Robertson, Atlantic Systems Guild
- Dave Thomas, Pragmatic Programmers

**INDUSTRY ADVISORY BOARD**

- Robert Cochran, Catalyst Software (chair)
- Annie Kuntzmann-Combelles, Q-Labs
- Enrique Draier, MAPA LatinAmerica
- Eric Horvitz, Microsoft Research
- David Hsiao, Cisco Systems
- Takaya Ishida, Mitsubishi Electric Corp.
- Dehua Ju, ASTI Shanghai
- Donna Kaspersen, Science Applications International
- Pavle Knaflic, Hermes SoftLab
- Wojtek Kozaczynski, Microsoft
- Tomoo Matsubara, Matsubara Consulting
- Masao Matsumoto, Univ. of Tsukuba
- Dorothy McKinney, Lockheed Martin Space Systems
- Stephen Mellor, Project Technology
- Susan Mickel, AgileTV
- Dave Moore, Vulcan Northwest
- Melissa Murphy, Sandia National Laboratories
- Kiyoh Nakamura, Fujitsu
- Grant Rule, Software Measurement Services
- Girish Seshagiri, Advanced Information Services
- Chandra Shekaran, Microsoft
- Martyn Thomas, Praxis
- Rob Thomsett, The Thomsett Company
- John Vu, The Boeing Company
- Simon Wright, Integrated Chipware
- Tsuneo Yamaura, Hitachi Software Engineering

**MAGAZINE OPERATIONS COMMITTEE**

- Jean Bacon (chair), Thomas J. Bergin, Pradip Bose,
- Doris L. Carver, George Cybenko, John C. Dill,
- Frank E. Ferrante, Robert E. Filman,
- Fouzan Golshani, David A. Grier
- Rajesh Gupta, Warren Harrison,
- Mahadev Satyanarayanan, Nigel Shadbolt,
- Francis Sullivan

**PUBLICATIONS BOARD**

- Rangachar Kasturi (chair), Jean Bacon,
- Laxmi Bhuyan, Mark Christensen,
- Thomas Keefe, Deependra Moitra,
- Steven L. Tanimoto, Anand Tripathi