

A Software Chasm: Software Engineering and Scientific Computing

Diane F. Kelly

... in which I oppose the domain-independent solutions proffered by the software engineering community and the resulting isolation of the scientific-computing community.

Some time ago, a chasm opened between the scientific-computing community and the software engineering community. Originally, computing meant scientific computing. Today, science and engineering applications are at the heart of software systems such as environmental monitoring systems, rocket guidance systems, safety studies for nuclear stations, and fuel injection systems. Failures of such health-, mission-, or safety-related systems have served as examples to promote the use of software engineering best practices. Yet, the bulk of the software engineering community's research is on anything but scientific-application software.



This chasm has many possible causes. Let's look at the impact of one particular contributor in industry.

Creating the chasm

In a 1997 paper, Iris Vessey described a shift in computing philosophy that occurred in the late 1960s.¹ She quoted George E. Forsyth, the ACM's president at that time, as stating that computer science was a field of its own, separate from the application domains. The result, as Vessey points out, was the insistence that anyone who wanted to be taken seriously in the field of computing, and software engineer-

ing, must develop domain-independent techniques, methods, and paradigms. The pressure was such that claims of broad applicability became commonplace. So, the bridge between software engineering and any application domain became a single massive structure that everyone had to use.

In industry, a management paradigm based on domain-independent software solutions took hold. A typical scenario runs as follows.

Consider a company that develops mostly engineering applications. Their IT division is organized according to required background knowledge—engineers and scientists write engineering applications, data specialists write database applications, business people write accounting and human-resources applications, and systems people look after hardware, operating systems, and compilers.

A management paradigm shift based on domain-independent software solutions causes a reorganization along the lines of, "all software is treated the same; all software developers are the same." The result is that a physicist is developing a timesheet program, and a chemical-dispersion program is left in a corner because the new maintainer, originally from the business side, is terrified to touch it. The customer divisions cry foul every time the IT division sends a software developer steeped

Continued on p. 118

Continued from p. 120

in new software engineering paradigms but ignorant of the needed engineering background. Software developers with engineering knowledge are hired away from the IT division into the engineering divisions. Over time, the IT division shrinks away.

The engineering-software developers are now working in customer-specific silos, physically and organizationally separated from each other. They have little opportunity for interaction and no organizational support mechanism. Each developer builds his or her own bridge across the chasm to the software engineering resources he or she needs or finds useful. Some do a very good job. However, their engineering-application managers often have a limited understanding of software—it's not their area of expertise. When something goes wrong, these managers grab the software engineering textbooks promoting domain-independent practices.

Unfortunately, the bridges that these developers have built aren't the domain-independent bridge blessed by all these textbooks. The result is often confusion, conflict, and imposed practices that don't improve software quality or the developer's life. The engineering-software developer and software engineer's lack of interest in each other's discipline is now firmly entrenched on both sides of the chasm.

Supporting the three types of scientific developers

Today, as more and more scientists work in their fields by sitting at a computer, the need to engage these people in software know-how is growing. We can identify three groups of scientific-software developer:

- the industrial developer working in his or her specific application domain,
- the scientific researcher, many of whom are employed in academia, and
- the student engineer or scientist who will eventually join one of the other two groups.

Greg Wilson from the University of

Toronto gave a recent seminar on software practices for academic scientific researchers.² In Wilson's experience, the average academic researcher developing software for research in his or her scientific domain is so far removed from the software engineering world that simple practices such as using a debugger and saving labeled copies (configuration management) are novel. Another colleague of mine commented that generations of graduate students in scientific fields have been using error-prone software development practices simply because no one has convinced them that there are better ways to do things.

In another example, response to a required software engineering course for third-year electrical- and computer-engineering students has been overwhelmingly negative. This has two major causes. First, the students fail to acknowledge the high probability of software being part of their future jobs. Second, the curriculum, based on standard software engineering textbooks, fails to offer solutions that are obviously geared to the type of work the students might undertake.

All three groups of developers need appropriate software engineering knowledge to support their work. Robert Glass, in his Practical Programmer column in *Communications of the ACM*, exhorted software researchers to identify domain-specific software development

methodologies.³ We need a mapping, he explained, between the methodologies that software engineers are developing and the problems that application-domain people are solving. As he pointed out, this task isn't easy. One step is to characterize the problem, or application, domains. Even if we shrink the characterization to scientific or engineering software, this step is still difficult.

In October 2000, an International Federation for Information Processing working group addressed the issue of the architecture of scientific software.⁴ They made important inroads in characterizing scientific software's structures and development environments. However, according to a recent private communication with this working group's chair, Morven Gentleman of Dalhousie University, there seems to be no interest in continuing this work. It seems that neither the scientific-computing community nor the software engineering community sees any interesting work going on across the chasm, enough to start building separate bridges. The working group did identify some useful planks to drop into the scientific-software development bridge, such as a common software architecture, around which many scientific applications are designed. This and other common practices developed by separate groups of industrial scientific-software developers need further investigation.

So, we're left with three groups of scientific-software developers taking different approaches to the software engineering chasm. The scientific community isn't applying the current solutions offered by the software engineering community, for whatever reason. Software engineers must realize that they can't be separated from the application domains they're supposed to be supporting. We must build bridges to join ideas, needs, and solutions on both sides of the chasm. 

Software engineers must realize that they can't be separated from the application domains they're supposed to be supporting.

References

1. I. Vessey, "Problems versus Solutions: The Role of the Application Domain in Software,"

Proc. 7th Workshop Empirical Studies of Programmers, ACM Press, 1997, pp. 233–240.

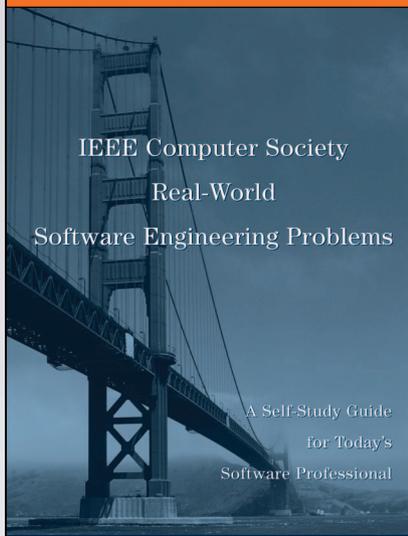
2. G. Wilson, "Where's the Real Bottleneck in Scientific Computing?" *Am. Scientist*, vol. 94, no. 1, 2006, p. 5; www.americanscientist.org/template/AssetDetail/assetid/48548.
3. R. Glass, "Matching Methodology to Problem Domain," *Comm. ACM*, vol. 47, no. 5, 2004, pp. 19–21.
4. R.F. Boisvert and P.T.P. Tang, eds., *The Architecture of Scientific Software*, Kluwer Academic, 2001.

Diane F. Kelly is an assistant professor in the Department of Mathematics and Computer Science at the Royal Military College of Canada. Contact her at kelly-d@rmc.ca.

Copyright and reprint permission: Copyright © 2007 by the Institute of Electrical and Electronics Engineers, Inc. All rights reserved. Abstracting is permitted with credit to the source. Libraries are permitted to photocopy beyond the limits of US copyright law for private use of patrons those post-1977 articles that carry a code at the bottom of the first page, provided the per-copy fee indicated in the code is paid through the Copyright Clearance Center, 222 Rosewood Dr., Danvers, MA 01923. For copying, reprint, or republication permission, write to Copyright and Permissions Dept., IEEE Publications Admin., 445 Hoes Ln., Piscataway, NJ 08855-1331.

IEEE Software (ISSN 0740-7459) is published bimonthly by the IEEE Computer Society. IEEE headquarters: Three Park Ave., 17th Floor, New York, NY 10016-5997. IEEE Computer Society Publications Office: 10662 Los Vaqueros Cir., PO Box 3014, Los Alamitos, CA 90720-1314; +1 714 821 8380; fax +1 714 821 4010. IEEE Computer Society headquarters: 1730 Massachusetts Ave. NW, Washington, DC 20036-1903. Subscription rates: IEEE Computer Society members get the lowest rate of US\$47 per year, which includes printed issues plus online access to all issues published since 1988. Go to www.computer.org/subscribe to order and for more information on other subscription prices. Back issues: \$20 for members, \$128 for nonmembers (plus shipping and handling). This magazine is available on microfiche.

Postmaster: Send undelivered copies and address changes to *IEEE Software*, Membership Processing Dept., IEEE Service Center, 445 Hoes Lane, Piscataway, NJ 08855-1331. Periodicals Postage Paid at New York, NY, and at additional mailing offices. Canadian GST #125634188. Canada Post Publications Mail Agreement Number 40013885. Return undeliverable Canadian addresses to PO Box 122, Niagara Falls, ON L2E 6S8, Canada. Printed in the USA.



IEEE Computer Society
Real-World
Software Engineering Problems

A Self-Study Guide
for Today's
Software Professional

Edited by
J. Fernando Naveda • Stephen B. Seidman

IEEE Computer Society Real-World Software Engineering Problems: A Self-Study Guide for Today's Software Professional



Publishers Since 1807




www.wiley.com/ieeecs

978-0-471-71051-6 • July 2006
310 pages • Paperback • \$60.00
A Wiley-IEEE Computer Society Press

To Order:
1-877-762-2974 **North America**
+ 44 (0) 1243 779 777 **Rest of World**

This book offers workable, real-world sample problems with solutions to help the professional software engineer solve common problems. In addition to its role as the definitive preparation guide of software engineering professionals for the IEEE Computer Society Certified Software Development Professional (CSDP) Certification Program, this resource also serves as an appropriate guide for graduate-level courses in software engineering or for professionals interested in sharpening or refreshing their skills.

15% off for
CS Members