Computational scientists across a multitude of disciplines write software, and software engineers have expounded upon software best practices for more than 40 years. In general, however, computational scientists don’t read software engineering literature and software engineers don’t concern themselves with the software that scientists write. This is the gap that was explored by participants in two workshops sponsored by IBM in Toronto, Canada.

The first workshop was led by a panel of six, each of whom expressed views on scientific software and concerns for dialog between software engineers and computational scientists. The second workshop consisted of six presentations of current research on software engineering topics relevant to scientists. The panelists and presenters represented a range of experience both in their years as practitioners and in their diverse backgrounds. Roughly half were from academic and research institutes, while the other half hailed from various industries.

Discussions in the two workshops pointed to missed opportunities for software engineering research to explore and provide novel and useful tools and approaches for scientists developing software.

First to emerge from the workshops was the delineation of two distinct types of software, both falling under the umbrella of “scientific software”:

- end-user application software, such as code written to model the distribution of microorganisms in the Alaskan gyre; and
- tools that provide support for scientists to express their models in code and execute their software solutions, including scientific libraries (such as BLAS/VTK) and program environments (such as Matlab).

Despite scientists feeling that software engineering has ignored them, scientists acknowledge that software has grown from the 1960s in ways that have changed substantially how a scientist works. Computations accomplished now were unheard of even a decade ago. The complexity of the computing carried out by scientists has increased dramatically.

Complexity of software in general hasn’t gone unnoticed by the software engineering community. Software engineers have developed processes and tools to manage complexity. Despite the fact that the workshop participants believe that such processes and tools are useful for software development, scientists often fail to take advantage of them. The complexity managed by these tools and processes is, for scientists, secondary in importance to the complexity inherent in their science. The scientific domain complexity embodied in scientific software won’t go away and manifests itself in various ways that software engineers aren’t addressing.

**Testing**

One manifestation is the lack of test oracles for scientific software. The problems to be solved by scientific software are always difficult. The reason the software exists is that the problem can’t be solved in other ways. Because the solution doesn’t already exist, there’s no accurate test oracle that can tell the scientist whether the answers are correct. This affects the whole activity of testing. Most software engineering testing techniques assume accurate oracles and the ability to realistically run large suites of tests and interpret the results. There’s a paucity of software engineering research into effective testing without oracles, and indeed on any code-testing technique specifically suited for scientific software.

**Code Review**

One panelist pointed out that testing results in “reviewing the output instead of reviewing the code.” Everyone agrees that code review is a good idea. However, it’s difficult to incorporate code review into work practices without the large overhead it commonly entails.

One panelist described her environment, in which a senior engineer reviews the work of subordinates. That is, the junior staff members bring in...
new ideas, while senior staff members impart their experiences. This is a common practice in areas such as structural engineering. Peer review of scientific software should be more widespread, including peer review of software used for academic research.

**Design**

Complexities in scientific software also contribute uncertainties in models and data, dependencies on interactions with computational methods, and difficulties in understanding what the computer has output. Software design for scientists has very special needs.

Expression of theoretical concepts in computational and, eventually, coded form involves several difficult transitions. A logical aid for the scientists is a way to express their solutions in a conceptually higher notation than code. Several communities have developed domain-specific languages but the problem is that they don't integrate together, leaving some scientists learning multiple languages.

Several communities have developed domain-specific languages but the problem is that they don't integrate together, leaving some scientists learning multiple languages.

**Process**

One panelist described software engineering as the endeavor to make a viable business out of developing and supporting software. For industrial applications more than academic applications, process becomes a contentious issue.

One of the panelists worked in a contractual situation where his institution hired an outside company to write scientific software. This software was in turn used to support the engineering needs of other customers. The immediate difficulty was in communicating their needs to the contractor, particularly in the timing of pertinent information. The contractors were using a waterfall development model, where the staged documentation fits well with contractual processes. Unfortunately, it doesn't fit well with the realities of scientific software development. After three years, the institution had “lots of documentation and nothing to sell.”

It's unfortunate that the waterfall software development model is the most commonly known development model, and probably one of the most studied in terms of its problems. For developing scientific software, other models are needed.

Using any software development process where the main goal of meeting delivery dates and budgetary limits is achieved by trading off quality and functionality won't work. Scientists need software that does everything it needs to do and doesn't lie to them.

One panelist described his company’s process as follows. A small group of multidisciplinary people develop the software. The software is “released” to engineers within the company for use in supporting external customers. The software remains in this internal release mode for approximately a year. By the time the software is released to external customers, it has undergone about a year of “real world” use and any problems found are fixed.

Software engineering has routinely stamped the development of scientific software as “ad hoc.” Yet scientific computation has a 60-year history, which would suggest that scientists have developed successful common practices over that time. Software engineering research needs
to identify and characterize these common practices.

**Tool Support**

A scientist’s goal and professional recognition comes through science, not through learning software tools. Immature, bloated, and buggy tools increase the scientist’s distrust of new technology. Researchers have cautioned scientists against being on the bleeding edge of new software technology: let others test it. The challenge to software engineers is to provide useful, reliable, and usable tools for scientists. This can only be accomplished by understanding how scientists work.

One presenter commented that tools are needed to tame the “incidentally” in developing scientific software. Science and mathematics are essentials for scientists, but software development is incidental—and incidentals get in the way.

Testing tools for scientists must support different perspectives on how scientific codes should behave. For example, testing tools must support floating-point arithmetic in ways that allow checking for different non-zero tolerances. Scientists don’t want fault tolerance in their code, they want fault intolerance: testing tools could help by causing the code to fail if a code fault causes an output that exhibits an inaccuracy greater than some limit.

Debuggers also came under fire. A common type of debugging problem in scientific software is the cause-effect chasm, in which evidence of the bug and the bug’s cause are widely separated. Debuggers don’t handle this well. Debuggers with visualization capabilities would be useful to scientists: time-series graphs or graphs based on other dependent variables would help visualize large sequences of data or internal variable values.

Tools that automatically extract equations from code are needed to support code reviews. Scientists could also use a “tweaking tool” to modify assumptions and trace the effect on the output.

Finally, participants identified user interfaces (UIs) as a messy problem. Scientists need a tool to automate UI generation that also enforces the separation between the UI and the calculations. Current tools hopelessly tangle code that handles graphical widgets with code that does calculations.

Currently, we lack mechanisms to pass along the experience and knowledge gathered by established scientists to the new scientists or the new software engineers helping them.

**Education**

Several participants expressed great concern over a general lack of necessary knowledge—specifically, scientists not understanding the complexity of the computer world they’re using, and software engineers ignorant of the scientific domain they’re supposed to support. As one participant put it, “Scientists typically have a lot of rigor in their work, but the same level of discipline fails to happen when they sit in front of a computer.” Another panel member said he’s finding that numerical calculations are being programmed by people who don’t understand them. And a recent article identifies “an oft-heard complaint on a Usenet newsgroup …: ‘why is 0.1 + 0.1 + 0.1 not equal to 0.3?’”

Currently, we lack mechanisms to pass along the experience and knowledge gathered by established scientists to the new scientists or the new software engineers helping them. Given the already cramped schedule for engineering and scientific curricula, there’s simply no time to thoroughly train the next generation in everything they should know. Choosing the most relevant topics to teach remains a difficult question. Education should not only make scientists aware of the appropriate software engineering techniques to make their work easier, but also make software engineers more savvy of scientific modeling, numerical methods, and the problems specific to scientific computing.

Scientists have long been writing code to support and explore their science. Workshop participants voiced myriad suggestions for future software research. Although scientists have been characterized as conservative in adopting new software tools and approaches, if they’re offered something that provides obvious advantages without compromising their code correctness, they’ll enthusiastically adopt it. Science is often concerned with pushing the boundaries of known knowledge, so it’s no surprise that scientists push the limits of all the tools they use. That is, they find new ways to break the tools. It also means they create new questions to answer.

The complexities inherent in scientific software are challenging. Continued discussions on scientific computing
comparing, computing will bridge the gap between scientists and software engineers and draw on the experience of both communities to address these problems.

Acknowledgments
In addition to Diane Kelly, the panelists and presenters were Kaska Kowalska (Maplesoft), Marc Kwee (Bruce Power), Morven Gentlemen (Dalhousie University), Alicia Grubb (University of Toronto), David Stubbs (Canada Masonry Design Center), Daniel Hook (Engineering Seismology Group), Mark Vidger (National Research Council), George Corliss (Marquette University), Jacques Carette (McMaster University), and Ned Nedialkov (McMaster University). IBM sponsored the workshop; support for individual presenters and panelists comes from the Natural Sciences and Engineering Research Council of Canada, the Royal Military College of Canada’s academic research program, and the presenters’ respective industries and universities.

References

Diane Kelly is an associate professor at the Royal Military College of Canada in Kingston, Ontario, Canada. Her research interests include anything related to software engineering and scientific software. Kelly has a PhD in software engineering from the Royal Military College of Canada. Contact her at kelly-d@rmc.ca.

Spencer Smith is an associate professor at McMaster University in Hamilton, Ontario, Canada. His research interests include the application of software engineering methodologies to improve the quality of scientific software. Smith has a PhD in civil engineering from McMaster University. Contact him at smiths@mcmaster.ca.

Nicholas Meng is a graduate student at Queen’s University in Kingston, Ontario, Canada. His research interests include the use of design recovery and symbolic execution in automating the numerical analysis of scientific software. Meng has a BS in mathematics and engineering from Queen’s University. Contact him at nickmeng@gmail.com.
PURPOSE: The IEEE Computer Society is the world’s largest association of computing professionals and is the leading provider of technical information in the field.

MEMBERSHIP: Members receive the monthly magazine Computer, discounts, and opportunities to serve (all activities are led by volunteer members). Membership is open to all IEEE members, affiliate society members, and others interested in the computer field.

COMPUTER SOCIETY WEBSITE: www.computer.org

OMBUDSMAN: To check membership status or report a change of address, call the IEEE Member Services toll-free number, +1 800 678 4333 (US) or +1 732 981 0060 (international). Direct all other Computer Society-related questions—magazine delivery or unresolved complaints—to help@computer.org.

CHAPTERS: Regular and student chapters worldwide provide the opportunity to interact with colleagues, hear technical experts, and serve the local professional community.

AVAILABLE INFORMATION: To obtain more information on any of the following, contact Customer Service at +1 714 821 8380 or +1 800 272 6657:

• Membership applications
• Publications catalog
• Draft standards and order forms
• Technical committee list
• Technical committee application
• Chapter start-up procedures
• Student scholarship information
• Volunteer leaders/staff directory
• IEEE senior member grade application (requires 10 years of practice and significant performance in five of those 10)

PUBLICATIONS AND ACTIVITIES

Computer: The flagship publication of the IEEE Computer Society, Computer, publishes peer-reviewed technical content that covers all aspects of computer science, computer engineering, technology, and applications.

Periodicals: The society publishes 13 magazines, 18 transactions, and one letters. Refer to membership application or request information as noted above.


Standards Working Groups: More than 150 groups produce IEEE standards used throughout the world.

Technical Committees: TCs provide professional interaction in more than 45 technical areas and directly influence computer engineering conferences and publications.

Conferences/Education: The society holds about 200 conferences each year and sponsors many educational activities, including computing science accreditation.

Certifications: The society offers two software developer certifications. For more information, visit www.computer.org/certification.

EXECUTIVE COMMITTEE

President: Sorel Reisman

President-Elect: John W. Walz

Past President: James D. Isaak

VP, Standards Activities: Roger U. Fujii

Secretary: Jon Rokne (2nd VP)

VP, Educational Activities: Elizabeth L. Burd

VP, Member & Geographic Activities: Rangachar Kasturi

VP, Publications: David Alan Grier (1st VP)

VP, Professional Activities: Paul K. Joannou

VP, Technical & Conference Activities: Paul R. Croll

Treasurer: James W. Moore, CSDP

2011–2012 IEEE Division VIII Director: Susan K. (Kathy) Land, CSDP

2010–2011 IEEE Division V Director: Michael R. Williams

2011 IEEE Division Director V Director-Elect: James W. Moore, CSDP

*Voting member of the Board of Governors

†Nonvoting member of the Board of Governors

BOARD OF GOVERNORS

Term Expiring 2011: Elisa Bertino, Jose Castillo-Velázquez, George V. Cybenko, Ann DeMarle, David S. Ebert, Hironori Kasahara, Steven L. Tanimoto

Term Expiring 2012: Elizabeth L. Burd, Thomas M. Conte, Frank E. Ferrante, Jean-Luc Gaudiot, Paul K. Joannou, Luis Kun, James W. Moore

Term Expiring 2013: Pierre Bourque, Dennis J. Frailey, Atsuhiro Goto, André Ivanov, Dejan S. Milojicic, Jane Chu Prey, Charlene (Chuck) Walrad

EXECUTIVE STAFF

Executive Director: Angela R. Burgess

Associate Executive Director: Director, Governance: Anne Marie Kelly

Director, Finance & Accounting: John Miller

Director, Information Technology & Services: Ray Kahn

Director, Membership Development: Violet S. Doan

Director, Products & Services: Evan Butterfield

Director, Sales & Marketing: Dick Price

COMPUTER SOCIETY OFFICES


Phone: +1 202 371 0101 • Fax: +1 202 728 9614

Email: hq.ofc@computer.org

Los Alamitos: 10662 Los Vaqueros Circle, Los Alamitos, CA 90720-1314

Phone: +1 714 821 8380

Email: help@computer.org

MEMBERSHIP & PUBLICATION ORDERS

Phone: +1 800 272 6657 • Fax: +1 714 821 4641 • Email: help@computer.org

Asia/Pacific: Watanabe Building, 1-4-2 Minami-Aoyama, Minato-ku, Tokyo 107-0062, Japan

Phone: +81 3 3408 3118 • Fax: +81 3 3408 3553

Email: tokyo.ofc@computer.org

IEEE OFFICERS

President: Moshe Kam

President-Elect: Gordon W. Day

Past President: Pedro A. Ray

Secretary: Roger D. Pollard

Treasurer: Harold L. Fleischer

President, Standards Association Board of Governors: Steven M. Mills

VP, Educational Activities: Tariq S. Durrani

VP, Member & Geographic Activities: Howard E. Michel

VP, Publication Services & Products: David A. Hodges

VP, Technical Activities: Donna L. Hudson

IEEE Division V Director: Michael R. Williams

IEEE Division VIII Director: Susan K. (Kathy) Land, CSDP

President, IEEE-USA: Ronald G. Jensen

revised 5 May 2011