Report: The Second International Workshop on Software Engineering for CSE

Held during the 2009 International Conference on Software Engineering, this workshop provided a venue for software engineering researchers to interact with CSE researchers and practitioners and further strengthen the evolving dialogue between them. This report offers a brief overview of the workshop's position papers and summarizes the breakout group discussions.

Although CSE software plays an important role in society, the software engineering community has historically given little attention to studying CSE's state-of-the-practice. In recent years, however, software engineering researchers and CSE software developers have begun creating a community for the mutual exchange of ideas and knowledge. Early support of this effort came from the International Conference on Software Engineering,[1] which held its first workshop on software engineering for high-performance computing in 2004; in the past two years, that workshop's focus expanded to include all types of CSE software. The motive for this expansion was the realization that, while supercomputer use entails unique challenges, most CSE software shares some common characteristics and issues, whether it's developed for use on a supercomputer or on a PC. Broadening the workshop to include all types of CSE software therefore made sense.

The benefits of a strong dialog between software engineering and CSE are clearly mutual. Because CSE software projects have unique characteristics, software engineers will have much to gain from and offer to the CSE community once key distinctions[2–4] between the two fields are properly understood:

- CSE projects often explore unknown science and thus many of the requirements (beyond the laws of nature) can't be known a priori and must emerge throughout the development process.
- Because CSE projects often investigate new scientific findings, the software's expected output is sometimes unknown, making it difficult or impossible to define test oracles. As a result, traditional software testing approaches are problematic.
- A CSE project's life cycle is likely to differ from traditional models. For example, in one workflow ("lone researcher"), a single scientist develops software to test a hypothesis and then discards the software. As another example, some projects can last 10 years or more and are in constant development throughout.
- CSE software sometimes requires extensive computing resources in terms of processing power for simulations or data throughput for data-intensive applications. These resource needs can require the development of complex software.

1521-9615/09/$26.00 © 2009 IEEE
Copublished by the IEEE CS and the AIP

Jeffrey C. Carver
University of Alabama

- Many CSE software developers are experts—often with a PhD in the underlying scientific or engineering domain—but have little formal training in software engineering tools and techniques. It's not uncommon for a single scientist to take on the role of software developer and then rely solely on the Internet to acquire relevant software development knowledge.

In this year's workshop, our focus was on discussing and understanding the development of two important types of CSE software. The first type is scientific or engineering software applications that aim to solve or provide insight into a specific scientific or engineering problem; examples include computational simulations and complex bioinformatics algorithms. The second type is software to support scientific or engineering applications, such as data management systems, workflow management systems, and data visualization systems.

Summary of Position Papers

The following summarizes the 2009 workshop's eight position papers. The full papers are available in the IEEExplore Digital Library (included in the ICSE proceedings); the workshop Web page (www.cs.ua.edu/~SECSE09) offers links to the proceedings and the presentation slides.

In "Testing for Trustworthiness in Scientific Software," Daniel Hook and Diane Kelly describe two problems that scientists face when testing their software: the lack of a test oracle and the many tests required to validate the code. Hook and Kelly maintain that, rather than fully testing the code for correctness, developers should instead test their code for trustworthiness using methods such as mutation testing.

In "Comparing Bioinformatics Software Development by Computer Scientists and Biologists: An Exploratory Study," Parmit Chilana, Carole Palmer, and Andrew Ko examined how bioinformatics software developers—including computer scientists and biologists—seek out the information they need to develop their software. To discover this, they interviewed developers and found that a developer's colleagues are an important source of information. Their paper also discusses how developers could employ better organization to find important information more easily.

In "Some Challenges Facing Software Engineers Developing Software for Scientists," Judith Segal describes two particular challenges that software engineers face when developing scientific software. First, scientists use their own software development models, which don't resemble traditional software engineering models. Further, the success of their models is affected by many (often unidentified) context variables. Second, community software can greatly impact some branches of science, yet developing this shared software entails many challenges that can derail its success.

In "Refactoring and the Evolution of Fortran," Jeffrey Overbey, Stas Negara, and Ralph Johnson describe how, as languages like Fortran evolve, they become more complex and often contain older and rarely used features. This complexity makes it difficult to use the language and to construct tools. However, tools that perform refactoring can eliminate complexity and make tool development easier. For example, removing global variables using the Photran tool will make a code compatible with Adaptive MPI, which performs load balancing.

In "An Empirical Characterization of Scientific Software Development Projects According to the Boehm and Turner Model: A Progress Report," Carlton Crabtree, A. Güne, Koru, Carolyn Seaman, and Hakan Erdogmus describe their planned research approach to identify characteristics of scientific software projects. They're conducting this research through a series of interviews with scientific software project teams. Their goal in identifying these characteristics is to help developers decide between using an agile development method or a plan-driven development method.

In "Integration Strategies for Computational Science & Engineering Software," Roscoe A. Bartlett explores the challenges developers face when they create CSE software by integrating software written by different expert groups. The CSE software domain poses some challenges for achieving this integration. In addition to exploring many of these challenges, Bartlett describes some integration approaches that have proven useful on the Trilinos project.

In "Barely Sufficient Software Engineering: 10 Practices to Improve Your CSE Software," Michael Heroux and James Willenbring note that CSE software developers are primarily focused on research or on advancing algorithms or modeling capability, rather than on formal software engineering. Because CSE developers lack training, resources, and time, they often fail to adopt appropriate software engineering techniques. Based on lessons learned from the Trilinos project, this paper presents 10 basic, lightweight practices that CSE developers can adopt to improve their software development process.

In "Injecting Software Architectural Constraints into Legacy Scientific Applications," David Woollard, Chris Mattmann, and Nenad Medvidovic note that, while using a formalized software architecture helps with maintenance, reuse, and evolution, many legacy CSE codes are written in languages—such as Fortran and C—that don't support architectural concepts. Such codes are therefore challenging to integrate into systems that use a formal architecture. To address this situation, they propose turning these legacy codes into components wrapped in architecturally aware interfaces. Developers can easily integrate these wrapped components into a system that uses architectural constructs, while also maintaining the legacy implementation's performance gains.

In "Reusability of FEM Software: A Program Family Approach," Wen Yu and Spencer Smith focus on program families, which are sets of related programs that share a common underlying functionality and code base, and are tailored for specific applications. As Yu and Smith describe, the program family approach is especially helpful for usability and maintenance. They also describe how this approach can benefit finite element method software and provide a proof of concept using a simple FEM program family for solving beam analysis problems.

In "Developing Scientific Applications Using Generative Programming," Ritu Arora, Purushotham Bangalore, and Marjan Mernik note that developers can use checkpointing to make CSE software resilient to failure when using large datasets and multiple resources. The problem, however, is that many codes don't have a built-in checkpointing feature. They describe how developers can use generative programming to reengineer an existing application in a nonintrusive manner. Using generative programming, developers can modify a code to include a checkpointing feature without changing the original code's functionality or performance.

In "How Do Scientists Develop and Use Scientific Software?" Jo Erskine Hannay, Hans Petter Langtangen, Carolyn MacLeod, Dietmar Pfahl, Janice Singer, and Greg Wilson describe the results of an online survey about how scientists develop and use software. Their survey produced almost 2,000 responses; the results indicate that scientists: consult peers for knowledge about software development and use; are more likely to use desktop and cluster machines than supercomputers; rely on software with a large user base; and believe testing is important but typically lack sufficient knowledge about it.

Finally, in "Preparing Scientists for Scalable Software Development," Valerie Maxville describes Australia's IVEC education program for eResearchers. CSE projects require developers who are highly competent in both the CSE domain and in software development. By observing multiple CSE projects, iVEC team members can identify which formal software engineering techniques will be useful in the CSE domain. iVEC currently trains CSE developers in the use of the message-passing interface (MPI) and is expanding its program to include additional lightweight and high-return techniques to improve CSE software development.

Breakout Groups

During the paper presentations, many interesting questions arose. To address them, we formed breakout groups around three key topics. Workshop attendees selected a group to join according to the question they wanted to discuss.

Facilitating Communication

The first question was: *How can scientists be effectively involved in software development and training?* One reason that CSE developers underutilize software engineering techniques is because of the communication difficulties between the software engineering and CSE communities.[2]

This breakout group focused on identifying methods to facilitate the flow of good ideas between the two communities. The group made several observations:

- Software engineering ideas and concepts must be translated into terms that CSE developers are familiar with and will understand.
- We need people who understand both the software engineering and the CSE domain.
- Software engineers can't make changes from the outside; the CSE team must know and trust them.
- Encouraging the use of software engineering for lone researchers (or small teams) is different than for large development teams.
- CSE teams won't be motivated to adapt software engineering until they encounter a vexing problem.
- The extreme programming practice of pair programming[5] is a possible way to bring software engineering to CSE projects.

Based on these observations, it's clear that there's still a communication chasm between software engineering researchers and the scientists who are developing software. To make advances in CSE, we must continue the effort to reduce this chasm.

Measuring Productivity

The second question was: *How can software engineering researchers measure their impact on the*

productivity of scientists? This question is key; both scientists and funding agencies care about *scientific productivity*, not software development productivity. Given this, it's important to determine how best to measure scientific productivity. First, however, we must determine how to define both productivity and the measurement time scale.

Software productivity metrics—such as lines of code per day—don't offer insight into scientific productivity. Typically, scientific productivity is measured by publications, citations, impact factors, and so on. Of course, such metrics take a long time to gather. Furthermore, it's not obvious that these metrics are actually good measures of scientific productivity (see Carlo Ghezzi's ICSE 2009 keynote; http://groups.google.com/group/keynote-discussion-carlo-ghezzi-icse09/?pli=1). In terms of measurement time scale, it's important to obtain concrete, valid results within a short timeframe. Funding agencies typically want to see documented results within six to 12 months. Also, when testing a new approach on a project, CSE developers must see some positive impact quickly to encourage them to continue using the approach.

The breakout group suggested three potentially fruitful ideas:

- *Indirect self-assessment*. Ask developers to identify tasks that they wish they could do but can't. Have the developers make such a list before introducing the new approach, three months later, and three months after that.
- *Hype*. If a new approach is effective, developers will recommend it to others, but it's not clear exactly how to measure this information.
- *Performance on an exam*. Measure performance on a suite of common programming tasks before and after training. However, this measure doesn't directly tie to scientific productivity for the same reason that using "best practices" doesn't necessarily improve scientific productivity.

To achieve a high level of relevance, it's imperative that software engineering researchers identify valid and reasonable methods for measuring scientific productivity.

Developing Community Software

The third question was: *What are the software development and use problems within scientific communities (as opposed to with individual researchers)?* Software developed by a community of developers or users is typically intended—at least in part—to enable communal data sharing. Often, such software is open source. The third breakout group focused its discussion on data sharing (primarily) and open source software. Several key issues emerged:

- Research groups clearly vary in their willingness to share data. Communities where the science is inherently multidisciplinary are more open to sharing data. Conversely, in communities where funding is in jeopardy, groups might be less willing to share data so as to maintain a competitive edge.
- Metadata and data misuse can be problematic. For example, to properly use data, it's important to know the data provenance: Does the laboratory that produced the data have a good reputation? What are the underlying assumptions of the model for which the data was produced?
- Agreeing on a data format can be problematic. A successful example here is Google Earth, which uses a simple data format to enable collaboration among geoscientists.
- Data ownership is important when producing papers. When many scientists work on the same problem, a paper's authors can run into (at least) the tens. There doesn't yet appear to be a consistent approach for dealing with this issue.
- In both North America and the UK (and possibly other countries not represented in this breakout group), funding institutions often require that the scientific software they fund be open source, which poses licensing complications. Problems can also arise when a system that's developed to address one particular problem isn't flexible enough to address the problems of other research groups.

To address these problems, the group concluded that it's important to first characterize the CSE communities to identify segments that behave differently. This idea follows on one raised in the 2008 workshop, where an initial list of characteristics was developed.[2] Among those characteristics were a scientific domain, the use of a high-performance computing platform, level of domain understanding, team size, and fault-tolerance level. During the 2009 workshop, we added the following attributes to that list: willingness to share data, funding stability, open source requirement, why a group is willing to be studied, the type of feedback provided to the team, and which software engineering techniques and tools are used.

Overall, the 2009 workshop was interesting and the discussions were lively. We look forward to holding more in the future. One question that wasn't fully addressed was whether there are

other venues where we might hold similar workshops and have greater participation from the CSE community. Among the suggestions so far are that we investigate a geosciences conference, the upcoming Society for Industrial and Applied Mathematics (SIAM) conferences on parallel processing and scientific computing (2010) and computational science (2011), and the International Conference on Computational Science (2010). If you're interested in more information or have venue suggestions, please contact me at carver@cs.ua.edu.

Acknowledgments

I'm grateful to the workshop attendees, who provided lively discussion and an interesting day. The breakout group notes were also an integral part of this report.

References

1. P.M. Johnson, "Workshop on Software Engineering for High Performance Computing Systems (HPCS) Applications," *Proc. 26th Int'l Conf. Software Eng.*, IEEE CS Press, 2004, pp. 772.
2. J.C. Carver, "First International Workshop on Software Engineering for Computational Science & Engineering," *Computing in Science & Eng.*, vol. 11, no. 2, 2009, pp. 7–11.
3. J.C. Carver, "SE-CSE 2008: The First International Workshop on Software Engineering for Computational Science and Engineering," *Proc. Int'l Conf. Software Eng. Workshop*, ACM Press, 2008, pp. 1071–1072; http://doi.acm.org/10.1145/1370175.1370252.
4. J.C. Carver et al., "Software Development Environments for Scientific and Engineering Software: A Series of Case Studies," *Proc. 29th Int'l Conf. Software Eng.*, IEEE CS Press, 2007, pp. 550–559.
5. K. Beck, *Extreme Programming Explained: Embrace Change*, Addison-Wesley, 2000.

Jeffrey C. Carver *is an assistant professor in the Department of Computer Science at the University of Alabama. His research interests include software engineering for computational science and engineering, empirical software engineering, and software process improvement. Carver has a PhD in computer science from the University of Maryland. He is a member of the IEEE Computer Society and the ACM. Contact him at carver@cs.ua.edu.*

IEEE computer society

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 WEB SITE: 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 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 14 magazines, 12 transactions, and one letters. Refer to membership application or request information as noted above.
Conference Proceedings & Books: Conference Publishing Services publishes more than 175 titles every year. CS Press publishes books in partnership with John Wiley & Sons.
Standards Working Groups: More than 150 groups produce IEEE standards used throughout the world.
Technical Committees: TCs provide professional interaction in over 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 credentials.
For more information, visit www.computer.org/certification.

IEEE
Celebrating 125 Years
of Engineering the Future

revised 1 May 2009

EXECUTIVE COMMITTEE

President: Susan K. (Kathy) Land, CSDP*
President-Elect: James D. Isaak*
Past President: Rangachar Kasturi*
Secretary: David A. Grier*
VP, Chapters Activities: Sattupathu V. Sankaran†
VP, Educational Activities: Alan Clements (2nd VP)*
VP, Professional Activities: James W. Moore†
VP, Publications: Sorel Reisman†
VP, Standards Activities: John Harauz†
VP, Technical & Conference Activities: John W. Walz (1st VP)*
Treasurer: Donald F. Shafer*
2008–2009 IEEE Division V Director: Deborah M. Cooper†
2009–2010 IEEE Division VIII Director: Stephen L. Diamond†
2009 IEEE Division V Director-Elect: Michael R. Williams†
***Computer* Editor in Chief:** Carl K. Chang†

** voting member of the Board of Governors † nonvoting member of the Board of Governors*

BOARD OF GOVERNORS

Term Expiring 2009: Van L. Eden; Robert Dupuis; Frank E. Ferrante; Roger U. Fujii; Ann Q. Gates, CSDP; Juan E. Gilbert; Don F. Shafer
Term Expiring 2010: André Ivanov; Phillip A. Laplante; Itaru Mimura; Jon G. Rokne; Christina M. Schober; Ann E.K. Sobel; Jeffrey M. Voas
Term Expiring 2011: Elisa Bertino, George V. Cybenko, Ann DeMarle, David S. Ebert, David A. Grier, Hironori Kasahara, Steven L. Tanimoto

EXECUTIVE STAFF

Executive Director: Angela R. Burgess
Director, Business & Product Development: Ann Vu
Director, Finance & Accounting: John Miller
Director, Governance, & Associate Executive Director: Anne Marie Kelly
Director, Information Technology & Services: Carl Scott
Director, Membership Development: Violet S. Doan
Director, Products & Services: Evan Butterfield
Director, Sales & Marketing: Dick Price

COMPUTER SOCIETY OFFICES

Washington, D.C.: 2001 L St., Ste. 700, Washington, D.C. 20036
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: John R. Vig
President-Elect: Pedro A. Ray
Past President: Lewis M. Terman
Secretary: Barry L. Shoop
Treasurer: Peter W. Staecker
VP, Educational Activities: Teofilo Ramos
VP, Publication Services & Products: Jon G. Rokne
VP, Membership & Geographic Activities: Joseph V. Lillie
President, Standards Association Board of Governors: W. Charlton Adams
VP, Technical Activities: Harold L. Flescher
IEEE Division V Director: Deborah M. Cooper
IEEE Division VIII Director: Stephen L. Diamond
President, IEEE-USA: Gordon W. Day

Next Board Meeting:
17 Nov. 2009, New Brunswick, NJ, USA