Editor: Rubin Landau, rubin@physics.oregonstate.edu

First International Workshop on Software Engineering for Computational Science & Engineering

By Jeffrey C. Carver

Computer codes are becoming increasingly important as researchers study complex scientific problems. These codes, to which graduate students and postdocs devote multiple people years, allow for scientific exploration that wasn't possible in earlier times. However, much of the effort spent developing code is wasted because good software engineering practices aren't followed. As difficult as it might be for some readers to believe, software engineers have spent considerable effort researching the most effective methods for planning, writing, testing, and documenting codes to allow them to be easier to debug and have a long, useful life.

In recognition of the general lack of exposure scientists have to software engineering and vice versa, a workshop was held during the 2008 International Conference on Software Engineering in Leipzig, Germany. The workshop's goal was to bring together researchers and practitioners from the software engineering and computational science and engineering (CS&E) communities to build a common understanding of the issues involved in the complex process of CS&E software development and identify common themes to pursue in future research.[1] Because cross-pollination between the communities is limited at best, there's a lack of effective software engineering techniques that specifically support CS&E software development. Software engineering researchers have developed effective techniques to support software development in other domains, so it's reasonable to determine whether the software engineering community can have similar success in developing techniques specifically for CS&E software. This workshop evolved from a series of workshops focused on software engineering for high-performance computing,[2–5] with the goal of broadening the scope to include all types of CS&E software.

In addition to the need for cross-pollination between the two communities, the conference organizers and attendees also believed that the following differences between the development of CS&E software and other types of software needed to be explored in more depth:

- The software must often implement sophisticated mathematical models and might be developed based on an executable specification, such as a series of Matlab equations.
- The software often explores unknown science, which makes it difficult or impossible to determine a concrete set of requirements a priori.
- The processes used for CS&E software development might differ significantly from traditional software development processes.
- Execution of CS&E software often requires powerful computing resources. Existing solutions that provide more computational power—clusters, supercomputers, grids—can be difficult to use.
- Successful CS&E software often revolves around its optimization to the machine architecture so that computations can be completed in a reasonable amount of time. The effort and resources involved in such optimization might exceed that required for the algorithm's initial development.

With these characteristics as a backdrop to the conversation, the workshop was convened with 14 attendees. Because the workshop was held during a software engineering conference, most of the attendees came from the software engineering community interspersed with a few representatives from the CS&E community. The workshop's papers and presentations are available at www.cs.ua.edu/~SECSE08. Four main themes emerged from the lively group discussions.

CS&E Software's Unique Characteristics

The first theme that arose during the discussions was whether CS&E software development really is different from other types of software development as we had assumed. An argument against the need for research into how to apply software engineering to developing CS&E software has been that traditional software engineering techniques would work for CS&E software if developers were properly trained. Partially in response to this argument, the workshop participants

Table 1. Context dimensions and potential values.

Dimension	Potential values
Use of high-performance computing machine	Yes / no
Type of data operated on	Floating point / strings / other
Focus on computation or throughput	Computation / throughput / both
Scientific domain	Weather forecasting / astrophysics / and so on
Domain understanding	High / medium / low / none
Team size	Number of team members
Purpose	Simulation / orchestration / exploratory / commercial innovation / other
Type of organization	Academic / corporate / government
Code distribution	Open source / commercial / other
Longevity	Number of years
Variation in end-user community	Large amount / small amount / no variation
Size	Number of lines of code
Processing types	Batch or interactive
Code evolution	Evolving / static
Level of fault tolerance	Very high / high / medium / low / none
Relationship between developers and users	Same group of people / some overlap / no overlap

developed a list of characteristics related to the developers, the development environment, and the users that differentiates CS&E software from other software.

In most cases, CS&E software developers have a scientific or engineering background and haven't received formal software engineering training. Most learn to develop software out of necessity rather than desire. As a result, typical CS&E developers view themselves primarily as scientists or engineers rather than software developers.

Within the CS&E environment, developers have to create unique types of software for a range of projects, from long-lived projects that exist for decades to those that are thrown away after one use, referred to as "Kleenex codes." Thus, each project imposes different constraints on development. Further, the goal of many CS&E software projects is to discover new science by exploring complex and ill-understood domains. These projects tend to involve a large quantity of numerical calculations, which will affect developers' choice of programming language. CS&E projects often support the search for new scientific results, so the requirements must evolve as the domain is better understood, in contrast to other development environments in which requirements evolve as a result of changing user needs or environments.

Many CS&E projects have to support a diverse user community, which can include casual users who are only interested in high-level results to power users who might go as far as to modify the code. In most cases, CS&E software has relatively simple user interfaces, although its execution is typically input driven. Therefore, developers have to ensure that user input doesn't create concurrency problems or deadlocks.

Appropriate Context Dimensions

The workshop attendees had very diverse backgrounds and experiences with many types of CS&E software. It quickly became clear that the positions each individual took during a discussion were colored by these experiences. Therefore, the group decided to enumerate as many of these dimensions as possible in the hope that a better understanding of a person's context would facilitate better discussion. In addition, these dimensions affect the decisions made during software planning and development, as well as the quality goals chosen for the projects. Table 1 contains a list of these dimensions along with their potential values.

Major Quality Goals

The definition of quality varied greatly among the workshop attendees. It was obvious that *performance* was an important goal for many CS&E projects, especially those that were targeted for execution on a supercomputer. However, other traditional software quality goals were less universally accepted or consistently defined. For example, the quality goal of *correctness* might seem like it would be universally relevant, but it wasn't. Different domains have different definitions—one was an answer's trustworthiness. In some domains, it's better for the software to crash or output no data rather than provide incorrect output. Another definition of correctness was software transparency, which lets developers or scientists make their own

Web Trends

For a brief look at current events, including program announcements and news items related to science and engineering, check out the following Web sites:

- Project helps prepare visually impaired children for computer science programs (www.nsf.gov/news/news_summ.jsp?cntn_id=112729&govDel=USNSF_51). The US National Science Foundation (NSF) has funded an initiative at the Rochester Institute of Technology that aims to increase the number of visually impaired students pursuing computer science degrees.
- Researchers store information at atom's nucleus (www.nsf.gov/news/news_summ.jsp?cntn_id=112538&govDel=USNSF_51). Scientists from Princeton University, Oxford University, and the Lawrence Berkeley National Laboratory describe their process of isolating a quantum bit while preserving its quantum information.
- CISE Pathways to Revitalized Undergraduate Computing Education (CPATH; www.nsf.gov/pubs/2009/nsf09528/nsf09528.html?govDel=USNSF_25). The NSF is soliciting proposals for developing student programs to further student competency in computational thinking. Deadline is 28 April 2009.
- Science & Engineering Visualization Challenge (http://www.nsf.gov/news/special_reports/scivis/index.jsp?id=challenge). The competition focuses on illustrations in science, engineering, and technology for education and journalistic purposes.

judgments about the output the software produces.

Another quality characteristic that was important in some cases was *testability*, although it wasn't always clear how to define the term. Sometimes software is tested by human inspection of a visualization result. In other cases, it's tested by performing sanity checks on known results—that is, using small examples with known results to increase confidence that the software will work on larger examples with unknown results.

Portability and *maintainability* are also important characteristics for some CS&E projects. For instance, if the software is intended to be productive for a long time, then such goals are important. The project will suffer if developers can't modify the software to keep up with advances in scientific knowledge or computer hardware.

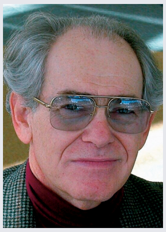
Finally, some of the attendees believed that *reusability* should become a more prominent goal. Currently, this goal isn't important for most CS&E projects. There are various, valid, and not-so-valid reasons why.

Crossing the Communication Chasm

Software engineering researchers and CS&E developers must address the large communication chasm that exists between them. Each group is partially responsible for the lack of communication and therefore can be part of the solution. Because most of the workshop participants came from the software engineering community, the first issue the group addressed was how software engineers could better reach out to computational scientists and engineers. The main suggestion was to conduct similar workshops at conferences that specifically target CS&E researchers

Observatoire Landau

Computational Scientific Thinking



By Rubin Landau, Department Editor

It's hard not to take notice when Carnegie Mellon University's computer science department—one of the country's premier CS departments—and Microsoft Research—the premier software company—start up an institute with the catchy title of Center for Computational Thinking (CCT; www.cs.cmu.edu/~CompThink/). With Jeanette Wing's paper on the subject (www.cs.cmu.edu/afs/cs/usr/wing/www/publications/Wing06.pdf) seemingly referenced by every third person in the computational science community, and Purdue University sponsoring a series of workshops (SECANT: Science Education in Computational Thinking; http://secant.cs.purdue.edu/) in which even physicists and biologists had views to contribute, I couldn't help but wonder if there might be something more here than just a catchy phrase (not to discount the importance of catchy phrases helping premier departments find success with grant proposals). I mean, isn't computational thinking what all of us reading this magazine have been doing for a living for years? Granted, after spending days debugging and formatting code, we might feel like we do more computation than thinking, but in the end, we do like to think that we are truly *Homo sapiens*.

According to the CCT, "Computational thinking is a way of solving problems, designing systems, and understanding human behavior that draws on concepts fundamental to computer science. To flourish in today's world, computational thinking has to be a fundamental part of the way people think and understand the world. Computational thinking means creating and making use of different levels of abstraction, to understand and solve problems more effectively; thinking algorithmically and with the ability to apply mathematical concepts such as induction to develop more efficient, fair, and secure solutions; understanding the consequences of scale, not only for reasons of efficiency but also for economic and social reasons."

Well, as someone who has been teaching computational physics and computational science for nearly two decades, I can't say that I disagree with these views, but I also can't say that they encapsulate my views of computational thinking. Of course, as a basic researcher and educator, my values, goals, prejudices, and measures of success differ from those of a computer scientist and so might be more accurately described as "computational scientific thinking." In fact, as a consequence of contributing to the Microsoft Research e-Science Workshop (http://research.microsoft.com/en-us/events/escience2008/) and planning an honors seminar on the subject, I've gathered some thoughts and present them here in the hopes of putting more science into computational thinking. I would say

- computational scientific thinking (CST) is using simulation and data processing to augment the scientific method's search for the truth and for the realities hidden within data and revealed by abstractions.
- concretely, as Figure A shows, CST is providing a coherent view of a natural system as the integration of data, theory, algorithmic model, and software implementation.

---

and developers. Another suggestion was to document instances in which software engineering researchers successfully addressed issues that are important to the CS&E community. These success stories will provide strong support for the benefits that can be gained from collaboration.

The group also noted that members of both communities must take steps outside of their comfort zones and try new things. In addition, they must eliminate their mutual distrust of one another and acknowledge the strengths they each provide. Further, software engineering researchers must understand that CS&E developers don't want their projects to become software engineering research projects. Software engineers must change the perception that all they have are solutions looking for problems. Rather, they need to listen to and learn from scientists' experiences prior to proposing solutions. Finally, when they find something that works, scientists and engineers should communicate this information to others who could benefit from it.

By bringing together a varied group of researchers and developers from the software engineering and CS&E communities, the workshop provided a forum for interesting discussions and knowledge exchange. The workshop attendees were enthusiastic about participating in a similar follow-up workshop at the 2009 International Conference on Software Engineering in Vancouver. One deficiency in the workshop was the underrepresentation by the CS&E community. To broaden its participation, I encourage your participation in this year's workshop, which will take place on 23 May 2009. More details can be found on the workshop's Web page (www.cs.ua.edu/~SECSE09) or by emailing me at carver@cs.ua.edu.

Acknowledgments

I thank the workshop participants whose comments and insights provided the material for this article. I also thank

- pragmatically, CST is learning the multiple disciplines needed to solve a problem and understanding them more deeply and more efficiently by understanding them in context. This entails learning the human and computer languages of multiple disciplines, respecting the values of these disciplines, and trading in good faith.
- CST practitioners gain control of their working environments by having the confidence to look at and understand the insides of computing black boxes and by having the courage to be nonexperts on some parts of a problem.
- computational scientific thinkers understand that it's more important to have the correct answer than the fastest answer and are willing to take on the hard work needed to obtain the correct answer.
- computational scientific thinkers recognize that there might be uncertainties and indeterminacies in computing the correct answer and that some mathematical colleagues might not think that a computed answer is an answer at all, yet the thinkers understand that moving beyond analytic solutions to approximate ones is often more realistic and accurate than elegant exact solutions.
- CST is the appeal of pursuing new science in complexity rather than developing different ways to view the same simple systems. It includes new subjects in science curricula, such as continuous media, nonlinear phenomena, space–time correlations, integral equations, wavelets, principle component analysis, (signal processing beyond Fourier), many-body theories, molecular dynamics, and imbued visualizations, for which computation is essential.

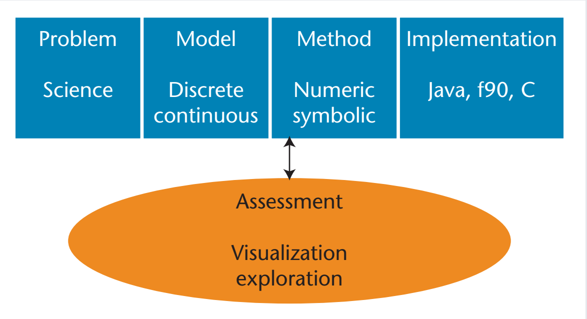


Figure A. Computational scientific thinking provides a coherent view of a natural system.

- in educational practice, CST might mean reversing the egalitarian trend of trying to make hard subjects more accessible by deemphasizing the importance of mathematics and abstractions. CST requires additional abstractions to understand and contribute to subjects such things as multidimensional representations of physical quantities and of data, and parallel and cloud computing languages.

I would appreciate hearing your thoughts on the subject—for future columns and to help improve my planned seminar. If you're interested in starting an Institute for Scientific Computational Thinking (something the US National Science Foundation's CISE Pathways to Revitalized Undergraduate Computing Education program might support), please let me know at rubin@science.oregonstate.edu.

Judith Segal for her comments on an early draft of this article.

References

1. J.C. Carver, "SE-CSE 2008: The First International Workshop on Software Engineering for Computational Science and Engineering," *Proc. 30th Int'l Conf. Software Eng. Companion Volume*, ACM Press, 2008, pp. 1071–1072.
2. J. Carver, "Third International Workshop on Software Engineering for High Performance Computing (HPC) Applications," *Proc. 29th Int'l Conf. Software Eng. Companion Volume*, IEEE CS Press, 2007, p. 147.
3. J.C. Carver, "Post-Workshop Report for the Third International Workshop on Software Engineering for High Performance Computing Applications (SE-HPC07)," *ACM Software Eng. Notes*, vol. 11, 2007, pp. 38–43.
4. P. Johnson, "Workshop on Software Engineering for High Performance Computing System (HPCS) Applications," *Proc. 26th Int'l Conf. Software Eng.*, IEEE CS Press, 2004, pp. 772–772.
5. P.M. Johnson, "Second International Workshop on Software Engineering for High Performance Computing System Applications," *Proc. 27th Int'l Conf. Software Eng.* (ICSE), ACM Press, 2005, pp. 683–683.

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.