Special Session: "Hands-On" Tutorial: Teaching Software Correctness with RESOLVE

Murali Sitaraman
Clemson University
School of Computing
Clemson, SC 29634
1-864-656-3444
murali@clemson.edu

Bruce W. Weide
The Ohio State University
Computer Science and Engineering
Columbus, OH 43210
1-614-292-1517
weide.1@osu.edu

SUMMARY
Program correctness is central to computing, with instructors striving to convey the importance of "getting it right" starting in CS1. Teaching this material carefully demands a uniform framework to specify, implement, and reason about software correctness. To make these ideas accessible to educators and students, the tutorial will use RESOLVE, an integrated specification and programming language with a toolset especially designed for building verified components. The tutorial will also discuss how to get students involved through hands-on activities with software construction and modular verification using a web-integrated environment that requires no software installation and that features a prototype "push-button" verifying compiler. The proposers have taught the ideas contained here using engaging pedagogical methods in introductory and advanced CS courses to thousands of students and dozens of educators over the past 20 years, and this SIGCSE tutorial will leverage that experience.

Categories and Subject Descriptors
D.2.4 [**Software/Program Verification**]: Including but not limited to the following – *programming by contract, class invariants, correctness proofs, formal methods.*
F.3.1 [**Specifying and Verifying and Reasoning about Programs**]: Including but not limited to – *assertions, invariants, pre- and post-conditions, specification techniques*.
F.4.1 [**Mathematical Logic**]: Including but not limited to – *proof theory, set theory, temporal logic*.

Keywords
Components, correctness proofs, programming by contract, specification, and verification.

1. OVERALL OBJECTIVE
Internalizing the close connections between mathematics and software engineering requires an understanding of the nature and use of mathematical modeling in software development. Students learn the connections between mathematics and physics because they practice making those connections. Do they learn similar connections for software engineering? Experience shows that students *can* reason rigorously and mathematically about software behavior—*when* they understand a few key principles and have the right tools. In the process, they realize the importance of the mathematics they learn for software development. Educators need compelling examples and assignments to make such connections, and often they are difficult to come up with, because the environments and backgrounds of students at different institutions vary widely. This session aims to provide a forum at SIGCSE where educators can share the joys of teaching mathematical reasoning principles throughout the computing curriculum using a framework especially designed for the purpose. The contents of the tutorial are suitable for teaching basic principles in introductory computer science, data structures, and discrete structures courses as well as more advanced topics in programming language and software engineering courses.

SIGCSE'14, March 5–8, 2014, Atlanta, Georgia, USA.
ACM 978-1-4503-2605-6/14/03.
http://dx.doi.org/10.1145/2538862.2538987

2. OUTLINE
Whatever doubts one might have had about the potential for automated verification a quarter-century and more ago [1, 2], automatically verified software can hardly be dismissed today. Empirical evidence now shows that verification conditions (VCs) implying software correctness are generally rather simple mathematically. In most situations, verification is "just" an enormous bookkeeping task involving VCs that modern theorem-provers often—yet still not always—can prove mechanically [3].

The tutorial will be of broad interest because it addresses how formal behavioral specifications, imperative programs, and supporting mathematics should be engineered for verification to be fully automated. The ideas apply in any language and under any verified software paradigm. Examples will deal with problems and solutions that illustrate how verified software can become closer to reality.

The tutorial will raise and address several issues, including the following. It will discuss the technical elements necessary for modular verification of software, i.e., proof of correctness one component at a time. It will illustrate the impact of macro-level and micro-level engineering of component specifications so that subsequent verification based on those specifications is amenable to automation. It will address how reusable mathematical developments can simplify specifications (e.g., by avoiding quantifiers) and ease automated verification. It will discuss necessary and sufficient annotations to be provided by engineers of object-based implementations to justify their correctness. It will consider verification in the presence of higher-order specifications, generic components, and components built by reusing yet other components. Verification of code involving computational and memory bounds will be among the topics of discussion.

A significant component of the tutorial will be "hands-on" activities with developing verified software [4, 5, 6, 7]. This will allow the

attendees to explore several "what if" scenarios, and among other ideas, understand how building software for verification naturally eases debugging and integration of component-based systems. The activities will be done through a web interface with no software installation, so attendees will be strongly encouraged to bring their laptops and participate directly in real-time on their own solutions to exercise problems.

The session will be organized as follows. The proposers will take the first 30 minutes to motivate and demonstrate how one might approach teaching rigorous reasoning about software correctness to undergraduate students. Following the introduction, about 15 minutes will be devoted to the attendees engaging in a simple exercise to reinforce their understanding using the RESOLVE web IDE. This will be followed by another 10 minutes of tutorial and a second exercise lasting about 10 minutes. Throughout the tutorial, the presenters will take clarification questions from the audience, while leaving more detailed questions to the end. The last few minutes will be devoted to seeking feedback from session participants on the format of the session, the choice of exercises, and other ways the tutorial could be improved for the future.

3. EXPECTATIONS

The intended audience is SIGCSE educators who teach courses ranging from an introductory sequence to ones that cover more advanced topics. The attendees will learn principles for developing verified software components, including the required close connections to mathematics, and understand how to engage students in exploring those principles for specification and verification through a web-integrated environment.

4. SUITABILITY FOR SPECIAL SESSION

Learning how to build correct software—software that behaves according to its specification—is an implicit or explicit learning outcome of most computing courses. There are no educational systems available today that support professors in communicating the necessary principles for developing correct software components. This tutorial will address the missing pieces. The presentation is intended as a tutorial supported with online materials and a web-integrated environment, and it fits the special session requirements. The proposers have been involved with SIGCSE for many years and have the qualifications necessary to make this an engaging and informative special session.

SIGCSE has routinely had a sizeable audience with interests in mathematical reasoning (and its use in related areas, such as programming languages and software engineering) as evidenced by panels (e.g., [8, 9]) and a continuing series of BoF sessions on mathematical thinking [10]. This tutorial will be of direct interest to that audience. It will also be of interest to attendees looking for innovative language alternatives to teach core CS principles.

Murali Sitaraman is Professor of Computer Science in the School of Computing at Clemson University. He directs the RESOLVE/Reusable Software Research Group (RSRG) at Clemson. His research spans a spectrum of foundational, practical, and educational topics in software engineering, and it has been supported by US NSF grants continuously for over 20 years. He has co-edited a book on *Foundations of Component-Based Systems* with Gary Leavens for Cambridge University Press. Together with members of his group, he has published over 100 papers in various forums. He has offered tutorials and workshops to educators on teaching mathematical reasoning at the ACM SIGCSE conference several times, including each of the last four years. Sitaraman has taught these ideas to graduate and undergraduate students for over 20 years.

Bruce W. Weide is Professor Emeritus of Computer Science and Engineering at Ohio State University. He and his colleagues in the RSRG at Ohio State have studied and taught design, specification, and verification of component-based software for over 30 years. Weide and his colleague, Tim Long, were awarded the IEEE Computer Society Computer Science and Engineering Undergraduate Teaching Award in 2000 "for innovative work in the content and pedagogy of introductory computer science education, linking research advances in software engineering with educational delivery of the material taught in the introductory courses."

5. ACKNOWLEDGMENTS

The members of our research groups contributed significantly to the ideas contained in this proposal. We also acknowledge NSF grants CCF-0811748, CCF-1161916, CCF-1162331, CNS-0745846, DUE-0942542, DUE-1022191, and DUE-1022941. Any opinions, findings, conclusions, or recommendations expressed here are those of the authors and do not necessarily reflect the views of the National Science Foundation.

6. REFERENCES

[1] DeMillo, R.A., Lipton, R.J., and Perlis, A.J. "Social Processes and Proofs of Theorems and Programs", *CACM 22*, 1979, 271-280.

[2] Fetzer, J.H., "Program Verification: The Very Idea", *CACM 31*, 1988, 1048-1063.

[3] Sitaraman, M., Adcock, B., Avigad, J., Bronish, D., Bucci, P., Frazier, D., Friedman, H.M., Harton, H., Heym, W., Kirschenbaum, J., Krone, J., Smith, H., and Weide, B.W., "Building a Push-Button RESOLVE Verifier: Progress and Challenges," *Formal Aspects of Computing 23*, 2011, 607-626.

[4] Cook, C.T., Harton, H.K., Smith, H., and Sitaraman, M., "Specification Engineering and Modular Verification Using a Web-Integrated Verifying Compiler," *Proc. 34th International Conference on Software Engineering*, IEEE/ACM, 2012, 1379-1382.

[5] Cook, C.T., Drachova, S. V., Sun, Y-S., Sitaraman, M., Carver, J., and Hollingsworth, K. E., "Specification and Reasoning in SE Projects Using a Web-IDE," *Proc. 26th Conference on Software Engineering Education and Training*, IEEE, 2013.

[6] Welch, D., Cook, C. T., Sun, Y-S., and Sitaraman, M., "A Web-Integrated Verifying Compiler for RESOLVE: A Research Perspective," *Proc. 7th India Software Engineering Conference*, ACM, February 2014.

[7] http://www.cs.clemson.edu/group/resolve (3 Sept 2013).

[8] Krone, J., Baldwin, D., Carver, J.C., Hollingsworth, J.E., Kumar, A., and Sitaraman, M., "Teaching Mathematical Reasoning Across the Curriculum", *Proc. 43rd ACM Technical Symposium on Computer Science Education*, ACM, 2012, 241-242.

[9] Gries, D., Marion, B., Henderson, P., Schwartz, D., "How Mathematical Thinking Enhances Computer Science Problem Solving", *Proc. 32nd ACM Technical Symposium on Computer Science Education*, ACM, 2001, 390-391.

[10] Baldwin, D., "Math-thinking-l – Mathematical reasoning in CS curricula", at: http://mail.geneseo.edu/mailman/listinfo/math-thinking-l/ (3 Sept 2013).