

The Importance of Mathematics to the Software Practitioner

Doug Baldwin and Peter B. Henderson

In the spring of 2000, Timothy Lethbridge released a study of the knowledge that practicing software professionals find important.¹ One of his claims is that practitioners find little use for mathematics in their jobs; they particularly feel that a significant disparity

exists between math's prominence in their educations and its actual day-to-day importance. In this column, we review evidence that mathematical reasoning increases a software developer's competence. We suggest that a resolution of the apparent conflict between this evidence and that gathered by Lethbridge lies in what it means to "properly" educate developers in mathematical reasoning.



exists between math's prominence in their educations and its actual day-to-day importance. In a subsequent Loyal Opposition column, Robert Glass discussed this finding, suggesting that educators should rethink the attention they devote to mathematics in computing curricula.²

On the contrary, we believe that properly educating students to use mathematical reasoning as a tool for software development makes them better professionals. We consider any use of the techniques, concepts, and processes of mathematics or logic to solve problems, whether that use is explicit or implicit, to be mathematical reasoning. As computer scientists and educators, we find such reasoning invaluable in our own

Math and educational success

The oldest indications come from studies of factors associated with students' success in learning computer science. Researchers have conducted a number of such studies since the early 1980s. All found a strong correlation between mathematical ability and success in introductory computer science courses. For example, in 1980, John Konvalina and his colleagues compared students who withdrew from a first computer science course to students who stayed.³ They found that the number of previous college math courses and total number of previous math courses (college and high school) were the statistically most significant differences between the groups, with students who stayed in the course having more math. In 2000, Brenda Wilson and Sharon Shrock correlated midterm exam scores in an introductory computer science course to 12 possible success predictors.⁴ They found that high school math background was the second-most important predictor. While both Konvalina and Wilson studied US students, another study

Continued on p. 110

Continued from p. 112

correlated various background factors to final exam scores for Irish students in an introductory programming course for noncomputing majors.⁵ They found a statistically significant positive correlation between final exam scores and scores on a secondary school “leaving exam” in mathematics.

Although none of these studies tried to prove that mathematical aptitude causes students to succeed in computer science courses, finding a strong correlation in multiple generations of students and in multiple educational systems certainly suggests some connection between mathematical ability and programming ability. Also, all these studies examined first programming courses; explicit use of math typically has little if any place in such courses and so cannot account for the correlation between math background and success.

Keith Devlin suggests that the connection between mathematical and programming ability lies in abstraction’s essential role in both math and software development.⁶ He argues that facility with abstract reasoning in mathematics also improves your ability to reason abstractly about software. Although no one has empirically tested this argument, it is certainly interesting.

Formal methods and programming success

Another strand of evidence lies in attempts to empirically validate the claims made for formal software development methods. For this discussion, we define “formal methods” as any use of mathematics or logic to specify or verify a software system’s features. Many claims made for formal methods remain untested and are probably exaggerated, but evidence exists that formal methods can help improve software quality. For example, Peter Larson and his colleagues describe a head-to-head comparison of the development of a trusted gateway system both with and without formal specifications.⁷

The system produced using formal specifications had fewer defects, ran faster, and had a simpler code structure. Shari Pfleeger and Les Hatton reviewed an air traffic control project where some parts were developed with formal methods and some without.⁸ They cautiously concluded that formal methods in conjunction with other techniques did improve software quality.

The benefits attributed to formalism in these studies were often due to simple uses of it, little more than using mathematics to state specifications or designs. Even this modest use of mathematics, coupled with otherwise traditional development practices, produced these benefits:

- Developers could identify important conditions early in design instead of when nearly complete code failed tests.
- Bugs introduced during coding were easier to detect in testing.
- The bugs were correctable with less drastic repairs.

So, although formal methods are not the silver bullet for all software evils, developers do appear to benefit from using mathematics, even in modest ways, alongside other tools.

Interestingly, some of the arguments against formal methods are that they are “too mathematical,” that only specially trained people can use them, and that providing this training

to most developers is too expensive. These are, of course, all reasons why software engineers need more, not less, mathematical education.

Applied, not isolated, math

On the basis of this evidence, we feel that an ability to think and communicate mathematically does benefit software developers. At the same time, we do not doubt Lethbridge’s finding that practicing developers find math relatively unimportant. How to explain this contradiction?

First, it is important to realize that what ranked low in Lethbridge’s study was relatively pure mathematics—calculus, differential equations, and so on. Mathematically based elements of computer science—for example, computational complexity and algorithm analysis, and formal methods—ranked around the middle of the surveyed topics. So, developers appear to value areas of computer science that draw heavily on mathematics more highly than they value mathematics in isolation from computing.

This leads to a second, more important, point: “mathematics in isolation from computing” is exactly what most undergraduate computing curricula teach. These curricula teach math in “related” (a word that students inevitably translate as “not central to computing”) courses in a separate department, and seldom if ever use math in software development or programming courses. It is no wonder that such treatment leaves students with little understanding that math might apply to computing problems, let alone with any practical ability to apply math. As the old saying goes, “If you have a hammer, everything looks like a nail”—to developers educated only to use “hammers,” very little looks as if it requires other tools. Yet those who can wield the other tools reap benefits.

Although formal methods are not the silver bullet for all software evils, developers do appear to benefit from using mathematics, even in modest ways, alongside other tools.

If we are right, then software education needs to better integrate math into the main computing curriculum and to more systematically apply math to computing problems.

Attempts to do this to parts of the curriculum exist (for example, both of us have developed mathematically integrated introductory sequences), as does at least one complete experimental curriculum.⁹ No one has yet compared graduates of such programs who are in the workforce to graduates of less mathematically integrated programs, but such a comparison would be quite useful. In the meantime, it is premature to conclude that mathematics is unimportant to software practitioners or that it should be reduced in undergraduate computing education. $\square$

Further Reading

See www.math-in-cs.org/relevance.html for further references to the ideas discussed in this column.

References

1. T. Lethbridge, "What Knowledge Is Important to a Software Professional?" *Computer*, vol. 33, no. 5, May 2000, pp. 44–50.
2. R.L. Glass, "A New Answer to 'How Important Is Mathematics to the Software Practitioner?'" *IEEE Software*, vol. 17, no. 6, Nov./Dec. 2000, pp. 135–136.
3. J. Konvalina, S. Wileman, and L.J. Stephens, "Math Proficiency: A Key to Success for Computer Science Students," *Comm. ACM*, vol. 26, no. 5, May 1983, pp. 377–382.
4. B. Wilson and S. Shrock, "Contributing to Success in an Introductory Computer Science Course: A Study of Twelve Factors," *Proc. 32nd SIGCSE Tech. Symp. Computer Science Education*, ACM Press, New York, 2001, pp. 184–188.
5. P. Byrne and G. Lyons, "The Effect of Student Attributes on Success in Programming," *Proc. 6th Ann. Conf. Innovation and Technology in Computer Science Education*, ACM Press, New York, 2001, pp. 49–52.
6. K. Devlin, "The Real Reason Why Software Engineers Need Math," *Comm. ACM*, vol. 44, no. 10, Oct. 2001, pp. 21–22.
7. P. Larson, J. Fitzgerald, and T. Brooks, "Applying Formal Specification in Industry," *IEEE Software*, vol. 13, no. 3, May/June 1996, pp. 48–56.
8. S. Pfleeger and L. Hatton, "Investigating the Influence of Formal Methods," *Computer*, vol. 30, no. 2, Feb. 1997, pp. 33–43.
9. A. Sobel, "Empirical Results of a Software Engineering Curriculum Incorporating Formal Methods," *Proc. 31st SIGCSE Tech. Symp. Computer Science Education, SIGCSE Bull.*, vol. 32, no. 1, Mar. 2000, pp. 157–161.

Doug Baldwin is an associate professor of computer science at the State University of New York at Geneseo, with interests in programming and programming methods, computer graphics, and computer science education. He was co-project director of a series of NSF grants that developed Geneseo's "threefold" introductory curriculum in computer science. He received his BS, MS, and PhD in computer science from Yale University. He is a member of the IEEE Computer Society, ACM, ACM SIGCSE, and ACM SIGSOFT. Contact him at the Dept. of Computer Science, SUNY Geneseo, 1 College Cir., Geneseo NY 14454; baldwin@geneseo.edu.

Peter B. Henderson is head of the Department of Computer Science and Software Engineering at Butler University. His interests include software engineering and computer science education. He previously was a faculty member in computer science at the State University of New York at Stony Brook. He received his BSEE and MSEE from Clarkson University and his PhD from Princeton University. Contact him at Butler Univ., 4600 Sunset Ave., Indianapolis, IN 46208-3485; phenders@butler.edu.

Copyright and reprint permission: Copyright © 2002 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.

Circulation: *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; (714) 821-8380; fax (714) 821-4010. IEEE Computer Society headquarters: 1730 Massachusetts Ave. NW, Washington, DC 20036-1903. Subscription rates: IEEE Computer Society members get the lowest rates and choice of media option—\$42/34/55 US print/electronic/combo; go to <http://computer.org/subscribe> to order and for more information on other subscription prices. Back issues: \$10 for members, \$20 for nonmembers. This magazine is available on microfiche.

Postmaster: Send undelivered copies and address changes to Circulation Dept., *IEEE Software*, PO Box 3014, Los Alamitos, CA 90720-1314. Periodicals Postage Paid at New York, NY, and at additional mailing offices. Canadian GST #125634188. Canada Post Publications Mail Product (Canadian Distribution) Sales Agreement Number 0487805. Printed in the USA.

IEEE Software

How to Reach Us

Writers

For detailed information on submitting articles, write for our Editorial Guidelines (software@computer.org) or access <http://computer.org/software/author.htm>.

Letters to the Editor

Send letters to

Letters Editor, *IEEE Software*
10662 Los Vaqueros Circle
Los Alamitos, CA 90720
dstrok@computer.org

Please provide an email address or daytime phone number with your letter.

On the Web

Access <http://computer.org/software> for information about *IEEE Software*.

Subscribe

Visit <http://computer.org/subscribe>.

Subscription Change of Address

Send change-of-address requests for magazine subscriptions to address.change@ieee.org. Be sure to specify *IEEE Software*.

Membership Change of Address

Send change-of-address requests for IEEE and Computer Society membership to member.services@ieee.org.

Missing or Damaged Copies

If you are missing an issue or you received a damaged copy, contact help@computer.org.

Reprints of Articles

For price information or to order reprints, send email to software@computer.org or fax +1 714 821 4010.

Reprint Permission

To obtain permission to reprint an article, contact William Hagen, IEEE Copyrights and Trademarks Manager, at whagen@ieee.org.