The *Communications* Web site, <http://cacm.acm.org>, features more than a dozen bloggers in the BLOG@CACM community. In each issue of *Communications*, we'll publish selected posts or excerpts.

Follow us on Twitter at <http://twitter.com/blogCACM>

DOI:10.1145/3148245

<http://cacm.acm.org/blogs/blog-cacm>

Building Tools to Help Students Learn to Program

Philip Guo summarizes his first three years of research into building tools to support those learning computer programming.

Philip Guo
Learning Programming at Scale

<http://bit.ly/2vMvEth>

August 8, 2017

My current research trajectory centers on what I call learning programming at scale. Decades of prior research have worked to improve how computer programming is taught in traditional K–12 and university classrooms, but the vast majority of people around the world—children in low-income areas, working adults with full-time jobs, the fast-growing population of older adults, and millions in developing countries—do not have access to high-quality classroom learning environments. Thus, the central question that drives my research is: *How can we better understand the millions of people from diverse backgrounds who are now learning programming online and then design scalable software to support their learning goals?* In my first three years as an assistant professor so far, I have approached this question from three main directions:

1. Understanding why and how people from diverse backgrounds are learning programming.

One critical prerequisite for improving how programming is taught is to understand why and how people are currently learning and what obstacles they face. To work toward this goal, I have been studying traditionally under-represented learner populations and non-traditional learning environments.

I recently studied how older adults aged 60 and over are learning programming.¹ I found that they were often motivated by age-related reasons such as keeping their brains challenged as they aged, making up for missed learning opportunities during their youth, connecting with younger family members, and improving their job prospects. They reported a variety of age-related frustrations such as a perceived decline in cognitive abilities, lack of social opportunities to interact with tutors and peers, and trouble dealing with constantly changing software technologies.

With Chris Parnin and his students, we studied the unique challenges faced by female programmers when seeking and providing help on the popular Stack Overflow question-and-answer website.⁸ We found five participation barriers that affected women more than men: 1) not being aware of certain features of the site, 2) not feeling qualified enough to chime in with questions and answers, 3) being intimidated by the large size of the online community, 4) discomfort from interacting with strangers online, and 5) fear of appearing like they are slacking on the job.

My student Jeremy Warner and I also studied the recent phenomenon of *hackathons* where college students gather for 24-to-36-hour periods to learn coding by creating software prototypes.³ We found that the time-limited format of hackathons generated excitement and focus and that learning occurred incidentally, opportunistically, and from peers. However, some students were discouraged from attending by perceptions of an overly competitive climate, an unwelcoming culture, and fears of not having enough prior experience.

Parmit Chilana and I identified and studied an emerging population of college students¹⁰ and professionals at technology companies⁹ who want to learn programming but do not actually need to write code for their jobs. We call these people *conversational programmers* since their main goal is to learn just enough about programming to be able to hold productive technical conversations with programmers.

2. Designing new kinds of programming environments to support learners.

Current programming environments are designed to maximize the productivity of professionals who are already experts. Instead, I've been creating new environments to address the unique challenges faced by novice programmers, which can hopefully ease their path to eventually becoming experts.

First, I built a series of tools to help novices overcome a fundamental barrier to learning programming: understanding what happens “under the hood” as the computer runs each line of source code. These tools are all built on top of the Python Tutor Web-based programming environment (<http://pythontutor.com/>) that I created in 2010.¹⁶ Python Tutor (despite its outdated name!) lets users write code in languages such as Python, Java, C, C++, JavaScript, TypeScript, and Ruby; it runs the user's code and automatically visualizes what the computer is doing step-by-step. So far, over 3.5 million people from over 180 countries have used Python Tutor to understand and debug their code.

I extended Python Tutor with a real-time collaborative mode called Codechella¹², which lets multiple users connect to the same visualization session and work together to solve programming problems and tutor one another. I followed up on these ideas with Codeopticon¹¹, a real-time activity monitoring dashboard that allows a single tutor to simultaneously watch dozens of people working on the Python Tutor website and jump in to tutor multiple learners at once. My student Hyeonsu Kang and I then morphed Python Tutor into Omnicode,⁶ a live programming environment that continually visualizes the entire history of all program values to give programmers a bird's-eye view of execution. With Chris Parnin and our student Ian Drosos, we created HappyFace,⁷ a medically inspired pain scale embedded into Python Tutor to let users self-report their frustration levels.

The second set of challenges that I tackle here relates to the fact that novices have trouble installing, configuring, and managing the complex array of software tools required to become productive as programmers. I coined

a term called “*command-line BS-ery*” to refer to these sources of extrinsic complexity that demoralize novices. To help remove these complexities, Jeremy Warner and I created CodePilot,² a programming environment that lets novices quickly get started with pair programming and test-driven development by integrating real-time collaborative coding, testing, bug reporting, and version control management into a single unified system. Similarly, my student Xiong Zhang and I created DS.js,⁵ which lets novices get started learning data science by writing code to analyze data directly on any webpage instead of needing to download datasets and configure analysis software on their own computers.

3. Designing new formats for programming-related instructional materials.

My third major research direction involves studying the shortcomings of existing formats for programming-related instructional materials and then designing new instructional formats that improve the user experience for both creators and consumers of those materials.

My students and I analyzed all of the discussion forum messages in a popular programming MOOC¹⁴ and also how people navigated a computer programming digital textbook.¹⁵ We found that people often wanted to discuss runtime code execution state but had lots of trouble doing so since forums are purely text-based. From these findings, we propose that a better discussion forum for learning programming should integrate automatically generated visualizations of execution state and enable inline annotations of source code and output.

My student Mitchell Gordon and I created Codepourri,¹³ a new tutorial format and crowdsourcing workflow that lets Python Tutor users work together to create step-by-step tutorials by directly annotating runtime code visualizations. Since there are far more learners than experts, using learners as a volunteer crowd of workers is a potentially more scalable way to create coding tutorials than relying solely on experts. We found that crowd-created tutorials for simple code were accu-

rate, informative, and even contained insights that expert instructors missed.

Back to eliminating command-line BS-ery, my student Alok Mysore and I created Torta⁴, a macOS app that allows users to create step-by-step tutorials that span multiple command-line and GUI applications by simply demonstrating the intended actions on their computers. The Torta system: a) automatically records a screencast video along with relevant operating system events, b) generates a new kind of mixed-media tutorial with the benefits of both video and text formats, and c) gives step-by-step feedback to people who are following the tutorial and automatically runs certain steps. ■

References

- Guo, P.J. Older Adults Learning Computer Programming: Motivations, Frustrations, and Design Opportunities, *CHI 2017* (Honorable Mention Paper Award)
- Warner, J., and Guo, P.J. CodePilot: Scaffolding End-to-End Collaborative Software Development for Novice Programmers, *CHI 2017*.
- Warner, J., and Guo, P.J. Hack.edu: Examining How College Hackathons Are Perceived By Student Attendees and Non-Attendees, *ICER 2017*.
- Mysore, A., and Guo, P.J. Torta: Generating Mixed-Media GUI and Command-Line App Tutorials Using Operating-System-Wide Activity Tracing, *UIST 2017*.
- Zhang, X., and Guo, P.J. DS.js: Turn Any Webpage into an Example-Centric Live Programming Environment for Learning Data Science, *UIST 2017* (Honorable Mention Paper Award)
- Kang, H., and Guo, P.J. Omnicode: A Novice-Oriented Live Programming Environment with Always-On Run-Time Value Visualizations, *UIST 2017*.
- Drosos, I., Guo, P.J., and Parnin, C. HappyFace: Identifying and Predicting Frustrating Obstacles for Learning Programming at Scale, *VL/HCC 2017*.
- Ford, D., Smith, J., Guo, P.J., and Parnin, C. Paradise Unplugged: Identifying Barriers for Female Participation on Stack Overflow, *FSE 2016*.
- Chilana, P.K., Singh, R., and Guo, P.J. Understanding Conversational Programmers: A Perspective from the Software Industry, *CHI 2016*.
- Chilana, P.K., Alcock, C., Dembla, S., Ho, A., Hurst, A., Armstrong, B., and Guo, P.J. Perceptions of Non-CS Majors in Intro Programming: The Rise of the Conversational Programmer, *VL/HCC 2015*.
- Guo, P.J. Codeopticon: Real-Time, One-To-Many Human Tutoring for Computer Programming, *UIST 2015*.
- Guo, P.J., White, J., and Zanelatto, R. Codechella: Multi-User Program Visualizations for Real-Time Tutoring and Collaborative Learning, *VL/HCC 2015*.
- Gordon, M., and Guo, P.J. Codepourri: Creating Visual Coding Tutorials Using A Volunteer Crowd Of Learners, *VL/HCC 2015*.
- Zhu, J., Warner, J., Gordon, M., White, J., Zanelatto, R., and Guo, P.J. Toward a Domain-Specific Visual Discussion Forum for Learning Computer Programming: An Empirical Study of a Popular MOOC Forum, *VL/HCC 2015*.
- Warner, J., Doorenbos, J., Miller, B.N., and Guo, P.J. How High School, College, and Online Students Differentially Engage with an Interactive Digital Textbook, *EDM 2015*.
- Guo, P.J. Online Python Tutor: Embeddable Web-Based Program Visualization for CS Education, *SIGCSE 2013*.

Philip Guo is an assistant professor of cognitive science at the University of California, San Diego. His research spans human-computer interaction, online learning, and computing education. Learn more at <http://pgbovine.net/>