Conversations with a Prominent Propagator: Philip Guo

David P. Bunde
Zack Butler
Christopher L. Hovey
Cynthia Taylor

Encouraging faculty to adopt new, high-impact teaching practices, tools, and curriculum in computer science (CS) undergraduate education requires intentional planning and sustained effort. This article is the next installment in the series of interviews with prominent propagators: members of the CS education community who have successfully spread pedagogical or curricular innovations. The goal is to capture knowledge and experience s that others can use to propagate their own teaching projects.

For this article, we interviewed Philip Guo, an Associate Professor of Cognitive Science and Computer Science & Engineering at the University of California San Diego. Dr. Guo's research examines how people learn computer programming, data science, and machine learning/AI [1, 3, 5–8], and he builds tools to help people better understand code and data [2, 4, 11]. He is known for creating Python Tutor — a website where users can run and visualize code in Python, Java, C, C++, and JavaScript — which has been used by over 20 million people worldwide. Philip is also known for his work on exposing and addressing the Hidden Curriculum [9, 10], which educates students on the unwritten rules, norms, and insider knowledge that are not taught in classes but are nonetheless essential for students to succeed in undergraduate education.

Below are highlights of the interview, which ran approximately an hour. The transcript has been edited for clarity and style.

Q: How did Python Tutor get started?

PG: Python Tutor started around 2010 as a side project when I was a grad student. It was really motivated by my informal experiences teaching people Python, drawing memory layout diagrams. It seemed like that was an intuitive thing to do to explain something. At that time, the web was getting good enough that you could start putting up these diagrams.

So I started with a very simple system, and I didn't intend for it to be anything big. It just kind of grew from there. I think that a lot of the growth was fueled by the fact that MOOCs, the massive open online courses, started up around 2012, and that Python happened to be a very popular language for teaching intro programming. Then Python had a big rise in the 2010s because of both the introductory curriculum and for data science and scripting and stuff. I expanded to other languages later on, and it's been 15 years or so, and it's still running. I think the most notable thing is it's really a one man show. It's unusual in that it's just me behind the curtain. And, so that's the Python Tutor project.

Q: Can you describe your Hidden Curriculum work?

PG: The project is called A Hidden Curriculum Guide to HCI and Design at UC San Diego. And this was led by one of my undergrads. The motivation was to create a really bottom-up curated guide to how to navigate through our major; how to get internships and jobs; and a lot of what we call hidden curriculum knowledge, which are things that are not taught formally in classes and things often that

2153-2184/2024/MonthOfPublication - ArticleNumber

http://dx.doi.org/10.1145/3706588

professors don't even know about. Because these are things that students pass to each other through word of mouth.

Usually students from more privileged backgrounds—from higher socioeconomic status, people who are not underrepresented minorities—grew up just knowing more of the stuff because they picked it up from their parents or from their friends, or they went to a prestigious private high school or a public school where lots of kids went to Ivy League colleges. So, they just kind of know about these sorts of things about networking and finding resources. Whereas a large percentage of our students are first-generation college students or underrepresented minority students or transfer students from local California community colleges. Many of those students come to our school without much knowledge about these behind-the-scenes things.

We try to capture that wisdom by interviewing a lot of students and putting together a Google Doc that is a one-stop shop. We can share it in a syllabus, or if students have questions for me then I can just email them the link. By definition, that project is student-led, because I feel like the undergraduate students have the best sense of how their peers think and talk. We describe it as sort of a peer mentoring guide. Ideally you would have a peer mentor go over it with you, but to scale that up, we just created this doc.

We've seen this at other schools, where there are ad hoc guides created by students. And our hope is to make it a bit more formal of a process so that other schools can adopt that idea and try to create their own local guides. The main thesis behind that project, which you can see in the ICER paper [10], is that these guides have to be very hyperlocal. If we created a "Here's how to be a generically good HCI or computer science student", it probably has good things, but when students read it, it just seems like it's on some official website. By definition, it has to be so general and cover all different kinds of schools and different kinds of institutions and populations that the advice would be high level. It would still be better than nothing. But I think the really interesting thing comes in when you make it local to your school, for two reasons. One is that you can put very local specific details in. And two, I think from a reader perspective, it feels more personal, like "Oh, somebody at my school in my department made this. It feels like it's for us rather than some generic PDF."

Q: How and why do you make your work directly adoptable by students, rather than being aimed at instructors?

PG: By targeting learners directly, you can have more impact. By definition, there are many more novices and learners than there are instructors and experts. That has always been sort of an appeal for me. From a methodological standpoint, if you are more of an education researcher or a learning science researcher, you think about teachers more. Bottom-up, interfacing with learners directly, has always been my philosophy.

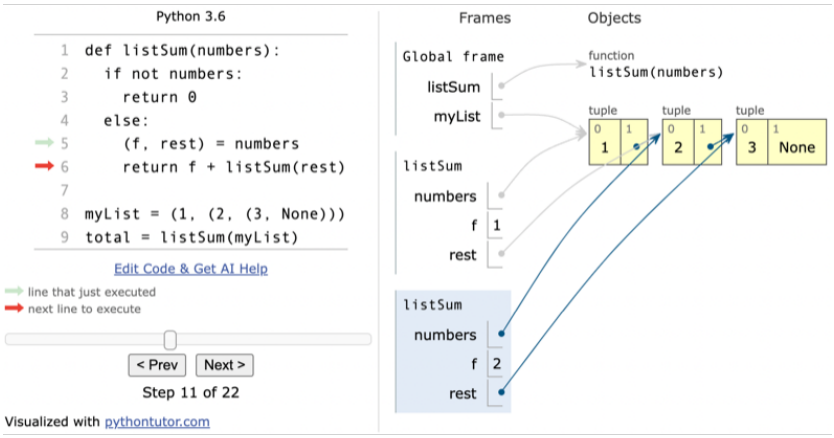
Python Tutor was originally meant as a teaching aid, but because it was online and it grew because of MOOCs and stuff, we found that most of the people using it are learners. That drove the development of the system too. Instructors would request certain kinds of features, things that make it easier to teach with, things that help you make exercises or make annotations or make lecture slides. I never really prioritized those features because they seemed very instructor-oriented. More important were student-oriented features like, "How do you make this work on more code and how do you make it accessible to more people?", because most people using it were novices who just pasted in their code and wanted it to work.

Q: How is propagation different when you are targeting learners as opposed to faculty?

PG: The traditional way is if you are a CS educator, usually as a professor, you have some tools you're building. You are trying to publicize that amongst faculty colleagues. I did not come from a CS education background, so that was not top of mind for me to do. By having something be online, it could grow more organically without all the in-person recruiting. To the extent that I did stuff, I think mostly it was fostering this organic growth.

The Python Tutor project was one of the first places you could go where you just visit a website, you don't have to log in or anything and you can just start writing code, running it, and visualizing it. I think that had immense appeal. Fifteen years ago, there were a few of these fledgling visualization tools and also IDEs for learners, but they all needed to be downloaded — like Java applets or downloadable apps. There's a much higher barrier to entry. Python Tutor saves all your code in the URL, and you can share whatever code you're typing. When you visualize it, the diagram essentially is just a combination of the code and what step you're in. You can have a few kilobytes of text in a URL. Keeping backward compatibility was a really big concern because I didn't want old links to break. If someone clicks on a forum post from five years ago, the URLs still work.

Having links online is powerful because someone else stumbles on it and clicks on the link and is like, “Oh, wow, that's pretty cool” and then they tell their friends about it. If well-known people in the Python community share it on social media, that gives you a boost. We got news coverage in various online forums, things like Hacker News and Slashdot and Reddit. A lot of these things are just a one-time boost, but it all adds up.



Suggested placement for image python-tutor-screenshot.pdf (scaled to approx 33% to fit within 2 columns of width) Caption: Visualization of a short Python program using Python Tutor

Q: As the only developer for Python Tutor, how do you make it sustainable?

PG: In general, I found that I didn't want anybody else on the critical path. It has to be really robust. That's a lot of pressure on someone else, and also, it's a lot of management overhead for myself. Instead

of trying to grow the number of people who work on it, I purposely continue limiting the scope of it. The scope hasn't really changed in 12 years; it largely does the same thing but it's more polished now. Having a user-centered motivation also helps keep the tool simple so that there aren't too many options and too many features.

In the beginning, I put my email address on the website and I had a GitHub issues page, and I was very receptive to seeing what people were doing with it. That was helpful early on to find the major bugs, but eventually it kind of stabilized. People were asking for similar sorts of things. I put "things I probably won't do," "things that I can do" and "known limitations" on the FAQ page. Then I took my email address off and I took down the GitHub issues page. I didn't have the mental bandwidth to handle those sorts of feature requests.

Q: How have you managed to fit this huge project into the reward structure in academia?

PG: A lot of the stuff I talk about with Python Tutor [2] is non-reproducible. It was an "n =1" case study with a very unique time and place and project. Whereas the Hidden Curriculum project [10], I feel like there's something generalizable in the model we set that other students and professors can take on and replicate. In my early career, Python Tutor really was fairly central in that I managed to get a few good publications on these variants and different tools we built upon it. We took advantage of the fact that there was a large user base. So if we were to build some kind of tool, we could deploy it and user test it in a very organic way. We could collect usage data.

The other way I used the Python Tutor platform was to run surveys. We had a few papers where we just put surveys or different activities on the website, and we'd basically use the site as a funnel to get users. There is a large audience of people learning programming in the wild, and you can get them from a lot of countries and different demographics. My NSF CAREER award was based on that line of work and there was an appeal there, because there's a platform and there's a lot of users and we have some papers on it. That story went really well together. With funding, the cliche goes that there's not much funding for infrastructure or software development. You really do have to spin into more of a research story.

In academia, things are valued based on novelty. Things that make for more robust software are less novel. If you describe Python Tutor in the abstract, there's nothing that novel about it because program visualization has been around for a long time. And the way that the memory diagrams are laid out is fairly standard stuff; there's nothing very radical about it. In fact, having it be less radical probably makes it more usable because it's what people expect to see. If you wanted to really innovate on program visualizations, then you might do something in VR like 3D visualizations and those would be cool for research papers, but it might be harder to adopt those. So there's this tension.

Q: How did you design Hidden Curriculum to make it replicable and sustainable?

PG: The Hidden Curriculum project was a very bottom-up thing. I started giving talks about how to thrive as an HCI or design undergraduate student. And I did that by collecting ideas from other students, frankly, because I'm 15 years removed from this, but they know how to do it. Then one of my undergraduate Teaching Assistants (TAs) and I started talking about things that are not taught in classes, and we formalized it more into this Google Doc guide project.

After we deployed this project and started seeing uptake, we thought maybe we could write about this because it feels like there's something generalizable here. We submitted it a few times and it didn't get in. Then we distilled down into this very learner-centered model of how other undergraduates could take this on, and what lessons generalize. As with a lot of these projects, it started with wanting to make something practical and then later, is there something more researchy we can extract from it? This goes back to your prior question about research versus product: I think the way to navigate those two is to think about "What can generalize? What is new here that people haven't talked about in the community?" And that's when you have the seeds of something researchy, even if it comes out of something more pragmatic.

Q: Do you think that part of the reason that your projects have been so popular is because you identified a practical problem that needs solving?

PG: Yeah, I think so. I have a lot of papers with students that are more the traditional model of "can we build a prototype and test it, or can we run a small study and validate something?" That works really well, but I feel like in order to get something adopted, it should start with real demand. I think it has more in common with doing a startup or doing product development.

When people are thinking about technology startups, you're really trying to find user needs and trying to find some corner of the market and you're not worrying about novelty. You're worrying about "Is this something in demand?" I encourage more projects like that in academia. There is a way to dovetail it with research as well, especially in computing education and HCI.

Q: What advice would you give somebody who is developing something with the idea of getting other people to use it?

PG: My top piece of advice, especially to someone in academia, is to just not worry about the research novelty. If you have to worry about that as well, then it just restricts what you can do. Just try one thing on the side with no worries about publications or grants or anything. That side project may turn into something bigger and if it doesn't, it's fine. In a sense, Python Tutor and the hidden curriculum were side projects: they were never meant to be main line research items. The challenge is bandwidth. If you are already having main projects and you're teaching and you're doing service work, how do you have one side project? That seems hard, but I think it is necessary.

Finding students who are similarly motivated by adoption is really important. Acknowledge that maybe you're not gonna get papers out of this, but if you get satisfaction out of people using it or seeing it and if that drives you, that is very powerful. And you end up being able to get papers out eventually, whether on a research spin off or on more of these sort of applied tracks like SIGCSE experience reports, which I think are great.

Q: How do you find motivated students?

PG: Finding the right person and scoping the project may be more important than finding a bunch of people. The hidden curriculum doesn't require a lot of labor, *per se*, if you have the right person. Whereas if you need an undergrad to build a piece of software that's very robust and deployable, that may be very hard because they're taking four classes and they wanna go do a summer internship at a company somewhere. A good undergrad project is something where they have some advantage

because they're an undergrad and it doesn't require an inordinate amount of work. That's sort of the sweet spot.

My student who did the Hidden Curriculum project transferred into our major very late. She felt like she was behind and she needed to really hustle to get her bearings straight. She could relate to new students who transferred in or might be first-generation students. If you can craft a project where the undergraduates actually have an advantage over graduate students or postdocs, that's really powerful and motivating. They're not just a lab assistant as an undergrad; they are actually doing the primary work and they are especially well-suited to do this work. It's really cool that those projects are better because the undergrads are doing them. That undergrad perspective is so important because as a professor or a graduate student, we may not see things that they see.

Q: What does success look like to you?

PG: I feel like success definitions should depend on the project. It could mean publishing a top tier paper or getting some paper award. It could also mean the students involved can use it to further their own career, such as my undergraduates using these publications to get into graduate school or to get internships or to get jobs. So I think that's a really powerful metric of success as well. Another metric is adoption. Every project doesn't have all of those.

For the Python Tutor project, success has been a lot of people using it and knowing about it. That's definitely something I've been proud of. I can leverage that in even more academic things, if you're writing grants or you are writing up something for your department. I think that especially in a very applied field, like in computing or HCI, usage numbers are good metrics. I don't want to say that everybody who's an academic needs to have software that's widely used. That bar I feel is not reasonable. But if someone does have that, I think it is a helpful thing. I think the community acknowledges that that is a good thing.

References

[1] Chung, A.M. and Guo, P.J. 2024. Perpetual Teaching Across Temporary Places: Conditions, Motivations, and Practices of Media Artists Teaching Computing Workshops. *Proceedings of the 2024 ACM Conference on International Computing Education Research - Volume 1* (New York, NY, USA, Aug. 2024), 374–388.

[2] Guo, P. 2021. Ten Million Users and Ten Years Later: Python Tutor's Design Guidelines for Building Scalable and Sustainable Research Software in Academia. *The 34th Annual ACM Symposium on User Interface Software and Technology* (New York, NY, USA, Oct. 2021), 1235–1251.

[3] Guo, P.J. 2018. Non-Native English Speakers Learning Computer Programming: Barriers, Desires, and Design Opportunities. *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems* (New York, NY, USA, Apr. 2018), 1–14.

[4] Guo, P.J. 2013. Online python tutor: embeddable web-based program visualization for cs education. *Proceeding of the 44th ACM technical symposium on Computer science education* (New York, NY, USA, Mar. 2013), 579–584.

[5] Kross, S. and Guo, P.J. 2019. Practitioners Teaching Data Science in Industry and Academia: Expectations, Workflows, and Challenges. *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems* (New York, NY, USA, May 2019), 1–14.

[6] Lau, S., Eldridge, J., Ellis, S., Fraenkel, A., Langlois, M., Rampure, S., Tiefenbruck, J. and Guo, P.J. 2022. The Challenges of Evolving Technical Courses at Scale: Four Case Studies of Updating Large Data Science Courses. *Proceedings of the Ninth ACM Conference on Learning @ Scale* (New York, NY, USA, Jun. 2022), 201–211.

[7] Lau, S. and Guo, P. 2023. From "Ban It Till We Understand It" to "Resistance is Futile": How University Programming Instructors Plan to Adapt as More Students Use AI Code Generation and Explanation Tools such as ChatGPT and GitHub Copilot. *Proceedings of the 2023 ACM Conference on International Computing Education Research - Volume 1* (New York, NY, USA, Sep. 2023), 106–121.

[8] Markel, J.M. and Guo, P.J. 2021. Inside the Mind of a CS Undergraduate TA: A Firsthand Account of Undergraduate Peer Tutoring in Computer Labs. *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education* (New York, NY, USA, Mar. 2021), 502–508.

[9] Nakai, K. and Guo, P.J. 2022. Scaling Up Access to the Hidden Curriculum: A Design Methodology for Creating Undergraduate Mentoring Guides. *Proceedings of the Ninth ACM Conference on Learning @ Scale* (New York, NY, USA, 2022), 413–417.

[10] Nakai, K. and Guo, P.J. 2023. Uncovering the Hidden Curriculum of University Computing Majors via Undergraduate-Written Mentoring Guides: A Learner-Centered Design Workflow. *Proceedings of the 2023 ACM Conference on International Computing Education Research - Volume 1* (New York, NY, USA, 2023), 63–77.

[11] Python Tutor - Python Online Compiler with Visual AI Help: *https://pythontutor.com/*. Accessed: 2024-10-04.

David P. Bunde
Knox College
2 E. South St
Galesburg, Illinois 61401 USA
dbunde@knox.edu

Zack Butler
Rochester Institute of Technology Rochester, NY 14623 USA
zjb@cs.rit.edu

Christopher L. Hovey
University of Colorado Boulder
1045 18th Street, UCB 315
Boulder, CO 80309
hoveyc@colorado.edu

Cynthia Taylor
Oberlin College
10 N Professor St
Oberlin OH, 44074
ctaylor@oberlin.edu