Conversations with a Prominent Propagator: Shriram Krishnamurthi

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 [2–4] with prominent propagators: members of the CS education community who have successfully spread pedagogical or curricular innovations. The goal is to capture knowledge and experiences that others can use to propagate their own teaching projects.

For this article, we interviewed Shriram Krishnamurthi, Professor of Computer Science at Brown University. He has done research in programming languages and CS Education for 30 years [6]. His work includes DrRacket (an IDE for Racket programming languages) [11]; Pyret (a free, functional programming language for education) [10] and Bootstrap (a high-school curriculum that incorporates CS into algebra, data science, physics and other subjects) [1]. He also co-developed an interactive version [8] of The Rust Book [9], the main reference for the Rust programming language. Their version includes a new conceptual model for Rust's ownership paradigm [5], the rules that govern how a Rust program manages memory.

Below are highlights of the interview, which ran approximately 1.5 hours. The transcript has been edited for clarity and style.

Q: How did you get into CS Education?

SK: I actually started in graduate school. Somebody asked my advisor, "What is the hardest problem you deal with?" and he realized it was not the technical problems, but the education question. We started an outreach program called Teach Scheme back in 1995. DrRacket, which was previously called PLT Scheme, was an outgrowth from that project. We brought in two teachers the first year, eight teachers the second year, 18 teachers the third year — a wonderful exponential curve. I did outreach nonstop from 1995 until about 2007, and then decided I wanted to take a bit of a break. Then I happened to meet Emmanuel Schanzer and started all over again.

Starting around 2011, I realized it was a research area, which is really a statement about me. Obviously, everyone else knew this. I didn't know what it meant to do CS education research. I'd hand out a survey and everyone said, "Oh, you did a great job. Wonderful!" Then I realized that I was not approaching education the way I do the rest of my research life. All the other work I do, I do related work, I define the research problem. I was like, "Why do I not treat education the same way?" Every course we teach is actually a research project. We go in, we do some things, we have some expectations of what'll work, what won't, and then we try to gather some data through assignments, through exams, whatever. And then we iterate.

We just happen to not treat this research area with the same rigor as we do anything else. We accept standards of rigor here that are ludicrous. And I could no longer reconcile that break in my head. And so I said, I've got to do this seriously.

2153-2184/2025/MonthOfPublication - ArticleNumber

http://dx.doi.org/10.1145/3778868

Q: One of your most recent projects is the Rust Book. Why did you choose to adapt an existing book?

SK: I want to be really clear, Will Crichton should get all the credit here. At some fundamental level, the reason I care about Rust is because for 40 years, the Programming Languages (PL) community has been yelling and screaming at systems programmers [about security]. They always had a credible excuse, and now they don't. Now that they're coming around, we have an obligation to help them and make things work. Will was well-connected to the Rust Book people and the Rust Book was already very popular, so why not try to improve something rather than build our own thing? That's the path to quickest adoption. We just innovated on one part. Anything we hadn't changed was exactly the Rust book. I think that was a pragmatically smart thing to do.

It only worked because Will had visibility and credibility in the community. There was also a technology thing: they had built the book in such a way that we could clone it and build it. And the really great thing was that Carol Nichols was willing to add a link to our copy, and that was the most amazing traffic driver you can imagine. So all of these things came together very nicely.

There's this old saying in Silicon Valley: you get one shot at eyeballs. Your users will try your website once, and if it doesn't work, they'll go away. Will and I both felt like there was a similar thing happening with Rust. Suddenly, it was taking up a lot of mind space. So where are people going to go? They're gonna go to the Rust website. From there, they're gonna go to the Rust book. If we can improve the quality of that educational experience, then the people who might otherwise have gone away will actually stay and struggle through it a little bit. And that'll be great.

We had a follow up paper this year where we studied the drop off graph of how people use the whole book [6], not just the ownership chapter [where we contribute a new conceptual model] [7]. And that has been useful because it's taught us a bunch of things about how to improve the book overall.

Q: What does that data acquisition look like, and how are you making those decisions of what to tweak and what to leave alone?

SK: For the Rust Book, for the ownership chapter in particular, we actually followed a pretty rigorous methodology that started by looking at a whole bunch of Stack Overflow searches to see what kinds of problems people are having. In parallel, we got 36 working programmers who were willing to be interviewed and debug a few programs live for us. That's what helped us understand what their difficulties were, and their misconceptions.

So, we had a multi-part approach to fixing it. One thing we did in rewriting the book was embed a whole bunch of quizzes and add analytics to the quizzes, and then use that to drive the quality of the text around the quizzes. Any failure to answer quizzes correctly is viewed as a failure of the prose and we'd go back and fix the prose.

Q: Collecting analytics does not seem very common in large scale projects. What analytics do you collect?

SK: We don't do logins, so our analytics are very lightweight. Actually Pyret is a more interesting case because in Pyret's case, the three main people — Joe Pollitz, Ben Lerner, and I — had been working on web security and privacy. So when we started building Pyret, we were like, "We have to be true to ourselves." In retrospect, it turned out to be a very good decision because now we have school districts who say, "You need to fill out this form about all the things you know about our students, before we can adopt your curriculum." And it's like, we don't know *anything* about your students. All we know in

Pyret's case is essentially browser headers. So how many different things, maybe not people, access a particular starter file. So we have starter files for Bootstrap Algebra, a starter file for Bootstrap Data Science. Now there's a starter file for Bootstrap AI, which I'm just launching this week. Just binning by starter files actually tells us a pretty good deal about who's using our curriculum without knowing anything about them.

Google Analytics gives us some geographic data. So we know we've got users in Ohio, users in Oklahoma. I expect to see a big lump from Rhode Island because people at Brown use it. We expect to see certain spikes. It confirms for us that our data are good, but we get just enough data to be able to say interesting things about Pyret usage. I don't want to know anything about you and I don't want to invade your privacy. It's a delicate dance.

Q: How did the connection between Bootstrap and Pyret come about?

SK: I was already part of the Bootstrap team. Emmanuel Schanzer did his undergrad at Cornell in CS and then worked for about a year at Microsoft. Then he decided his real calling was to be a teacher. As he saw his students struggling in math, he realized, "I can make this math curriculum much more lively if I could use some programming in it, but I've got to use functional programming because I don't want to do programming that violates the rules of mathematics."

So he's like, "When I was in college, I took this course from Greg Morrisett where he taught us this thing called Racket. So maybe I should download DrRacket and have my students play with that." So he starts getting his students to play with it, and it's going very successfully for him. It also turns out Emmanuel, by pure coincidence, is from Providence. He visited his dad in Providence and was taking the train back to Boston, and Matthias Felleisen happened to be on the same train. Matthias sees a laptop screen open with DrRacket on the screen, and walks over to Emmanuel and says, "That's the Racket logo." He's like, "Yes, it is. What do you know about Racket?" Matthias is like, "Well, I built it." That was the conversation starter.

So, Matthias sent Emmanuel to talk to Kathi and me and said, "These are the people in the Racket team that are doing education. You should go talk to them." That was the way the trio got together in 2007. By the time we invented Pyret, Kathy and I were deeply embedded in Bootstrap already. It was actually more like I forced it on Emmanuel.

Public transit, man, being able to see other people's screens — you never know what comes of it.

Q: How important do you think it is to build a textbook, curriculum, and/or assignments around an IDE in getting people to adopt?

SK: We've always said it's a three legged stool for stability. We control the educational materials, typically a book. We control the language. We control the IDE. Everything is interdependent on everything else, and everything is co-designed. If the error messages suck, I can just go fix them. I don't have to be like, "Oh, please run my version of some random thing at some URL." By controlling the whole experience, we can get a much better pedagogic outcome.

Q: In what ways have you experienced the tension between fidelity of adoption versus letting users adapt it?

SK: Nothing is ever adopted well. It's just embarrassing. I think it's a sliding scale between fidelity and quantity. You really have to decide for each thing where you want to be on those scales. Roughly speaking, fidelity is inversely proportional to quantity of adoption. So you have to decide where you want to hold your nose and where you don't. What do you consider a win? I would never put out a curriculum with my name on it that I don't feel really good about; if they're using even 20% of that as opposed to what they're currently doing, like I'm okay with that.

Q: What motivates you to continue work on long-term projects like DrRacket and Pyret?

SK: It's been super fun. Doing things outside the linguistic mainstream is always a fraught experience; it requires a significant amount of self-belief and willingness to see a long-term game. It helps to have a clear vision and a 25-year plan. In a given year, you're not going to make more than 4% progress and there's no point in getting stressed out about it. You never know when people suddenly get excited about something. You're like, "I have pride in the product and I'm just gonna make this thing amazing." Then, if and when people eventually show up and try it out, they're like, "Whoa, this is pretty mature. What's going on here?"

I've had that experience multiple times, which is useful because my students are always experiencing it for the first time. I've sort of sat them by the fireside and said, "Listen, young ones, let me tell you about the time we did this and that, and every time it paid off. But we had to just stick with it for several years."

Q: What has been your definition of success for a project?

SK: Oh, I don't think there is one. Sometimes the definition is just like large N, and sometimes it's the impact of ideas because ultimately I am a scientist. Why do I build software? Because I want people to download and run it. Why do I write papers? Because I want to get ideas out there that other people might be able to adopt. And so I am happy to recognize there are many, many paths to impact. You never know which thing people are going to pick up on, but you increase the likelihood of impact if you hit all the channels. Because there's so much randomness in the universe, you can't predict these things. You gotta pick all of them. So it involves foundational, theoretical work, applied work, evaluated work as well as software.

Q: What advice would you give someone who wants to get started in CS Ed?

SK: Ben Shapiro and I created this thing [CERAMICS] [12] where anybody in the community who wants advice on how to get into CSED research can talk to us. It's particularly designed for junior faculty, either you're a CSED person who's now on your own, you don't have an advisor, the cocoon has been taken away, or maybe you're in some other field of CS. You're a systems person and you're like, "is there some CS ed research I could be doing? Now I'm an educator, how do I do this?" We want to teach you how to do that.

Q: How can people have the most impact?

SK: I would say write fewer papers, write more code, make software that actually works that other people can use. You're much more likely to have impact. Build stuff of value. You build stuff and that's

how you grow community. That's how you get respect; people download and use your stuff if you can build something of utility.

The other thing I would say is that I don't write a paper till the system's been out for two to three years. And I think CS Ed in particular has this disease where you barely iterate on something once and you rush off to write a paper. Every single time I have waited and written a paper after multiple years and multiple iterations, it's been such a rich paper. In some cases the paper just writes itself. When you read it, you can tell that real work went into it. I don't like future work sections, right? Future work is always more of the same. Basically, here's my next paper. But a real discussion section: you can tell when you read a paper and somebody's really done the work. It's a discussion and it really says, we tried this, this didn't work. We tried that. Here's the thing to try and here's why to try it. People are terrified about saying what their next thing is because they're so afraid of being scooped. If you worked on something for three years before you wrote the paper, nobody's gonna be able to scoop you. You can write with confidence. You can do good science because nobody can catch up with you.

Q: How can researchers most effectively get others to adopt their innovations?

SK: I have one principle about adoption. My somewhat cynical take is that people are lazy. It's not a bad thing inherently, but there's benevolent laziness and malign laziness. The benevolent laziness is only making changes when they make sense, the malign laziness is I don't feel like learning a new thing. That take means that I have a strategy for the things that I produce. The assumption is that there is a professor who has their slides and they're never gonna change them. So instead you build the supplementary tool. Asking the professor to change anything in that class - never gonna happen. But telling the professor, “here's a tool and you can just sit your students in front of it and magically they're going to get better” – that's why something like Python Tutor wins. It's just a URL. So my very cynical takeaway is that my most successful things have been things that require nobody to actually change anything.

And that's kind of sad because we can't make progress unless we're willing to significantly change. I will say K-12 [pre-college] teachers have been more open to adoption that way than college professors, which is a damning indictment of our business.

Q: K-12 teachers think their job is to be teachers.

SK: Yeah. College professors think their job is to be the resident genius. K-12 teachers have all sorts of constraints, but they're still more willing to try something. And if you can make a case, they're willing to learn something. In the US there is also this whole mechanism of [K-12] professional development. But there is nothing in my job that says, “Hey, Shriram, you might wanna learn a new thing.” There's no time off for it. So when is that curricular innovation supposed to happen? How is it supposed to happen? In our department, we put in a mechanism to deal with this. You could propose changes and get paid partly in the summer.

Q: How do you entice people into the projects where they are going to have to do some work?

SK: People will reach out to you in those cases, a small number of people, but they'll reach out. They're like, “wow, this is really cool. What do I have to do?” And then what you have to say is, “don't worry. My student and I will support you. We're glad you want to try this out. Give us some examples of things you want to try out. We'll build an example prototype for you.” You have to meet them more than halfway.

It really is customer service. There's a principle I learned very early that I teach all of my students. The best thing that can ever happen to your software is a bug report, because it means somebody ran it enough to find a situation you had not covered in your testing and cared enough to tell you about it. Most buggy software, the user just deletes it and moves on. Instead, they stopped. They looked up how to report it, they reported it. I don't care what deadlines you have, you drop everything and say, “I'm on it. Thank you. I will fix it for you.”

And fix it. No matter how stupid or trivial it is, you have to give them that attention. They're like, “wow, these people care.” And now, they reciprocally care. I had this happen once. It was in grad school, I pushed out a release of DrScheme [the predecessor to DrRacket] where I had done something really stupid. It turned out if you wrote an expression that was nested three levels deep, everything crashed. We put out the release at night. The next morning we come in and there's like 500 bug reports. I fixed it within an hour, here's the patch, here you go. And they're like, “oh, that wasn't bad. They fixed it right away.” And then they forgave us. If you look at it as “I'm onto my next paper,” this person is a nuisance. If you think of it as “I'm putting out a product that I'm proud of,” then you're like, “What can I do to help you?” The papers can wait. And now you've got the user and you got a richer story. You got a new use case, your paper just became a whole lot better. You have to have that attitude.

References

[1] Bootstrap: *http://www.bootstrapworld.org/*. Accessed: 2025-08-21.

[2] Bunde, D.P., Butler, Z., Hovey, C.L. and Taylor, C. 2024. CONVERSATIONS: Conversation with a Prominent Propagator: Susan H. Rodger. *ACM Inroads*. 15, 4 (2024), 9–13. https://doi.org/10.1145/3699720.

[3] Bunde, D.P., Butler, Z., Hovey, C.L. and Taylor, C. 2025. CONVERSATIONS: Conversations with a Prominent Propagator: Amy Ko. *ACM Inroads*. 16, 3 (2025), 26–30. https://doi.org/10.1145/3756325.

[4] Bunde, D.P., Butler, Z., Hovey, C.L. and Taylor, C. 2025. CONVERSATIONS: Conversations with a Prominent Propagator: Philip Guo. *ACM Inroads*. 16, 1 (2025), 17–21. https://doi.org/10.1145/3706588.

[5] Crichton, W., Gray, G. and Krishnamurthi, S. 2023. A Grounded Conceptual Model for Ownership Types in Rust. *Proc. ACM Program. Lang.* 7, OOPSLA2 (Oct. 2023). https://doi.org/10.1145/3622841.

[6] Crichton, W. and Krishnamurthi, S. 2024. Profiling Programming Language Learning. *Proc. ACM Program. Lang.* 8, OOPSLA1 (2024). https://doi.org/10.1145/3649812.

[7] Crichton, W., Krishnamurthi, S. and Gray, G. 2025. Chapter 4: Understanding Ownership. *The Rust Programming Language - Experimental Version*. S. Klabnik, C. Nichols, and C. Krycho, eds. https://rust-book.cs.brown.edu/ch04-00-understanding-ownership.html.

[8] Crichton, W., Krishnamurthi, S. and Gray, G. 2025. The Rust Programming Language: Experimental Fork. *The Rust Programming Language - Experimental Version*. S. Klabnik, C. Nichols, and C. Krycho, eds. https://rust-book.cs.brown.edu/.

[9] Klabnik, S., Nichols, C. and Krycho, C. 2025. *The Rust Programming Language*. No Starch Press. https://doc.rust-lang.org/book/.

[10] Pyret: *https://pyret.org/@|full-uri|*. Accessed: 2025-08-21.

[11] Racket, the Programming Language: *https://www.racket-lang.org/*. Accessed: 2025-08-21.

[12] Shapiro, R.B. and Krishnamurthi, S. CERAMICS: Computing Education Research Advancing Methods for Curricula and Systems.

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