

Teaching Statement

Siddhartha Prasad

I believe I am uniquely well positioned to prepare students for the changing practice of software engineering because of my background in three distinct but complementary areas. My PhD **research focuses on specifying and reasoning about software behavior**, skills becoming increasingly important as generative AI shifts the center of gravity of programming from writing implementations toward specifying intent and evaluating what gets built. My earlier **research in computing education** investigated the misconceptions students develop when learning programming and logic, and translated those insights into educational tools now used in courses at multiple universities. Finally, my **years as a professional software engineer** give me firsthand experience of the profession and insight into what makes engineers effective beyond just their ability to write code. Together, these experiences inform both what I aim to teach students about software and how I help them learn it.

Teaching Philosophy

The joy of teaching, for me, comes from the dialogue that unfolds as students and I work through ideas together. Students who are empowered to contribute to classroom discourse give instructors insight into their understanding, what they find valuable, and how they reflect on their own learning.

Making it safe to be wrong. Teaching as dialogue only works when students feel comfortable asking “silly” questions, admitting when they are confused, and giving a “wrong” answer. As a student, I saw how ridicule from teachers and classmates could make staying confused feel safer than asking a question. Later, as a TA, I saw similar hesitation during office hours: students were reluctant to ask questions in front of their peers for fear of sounding foolish. An instructor has a responsibility to reduce this social cost, and it goes beyond just saying “there are no dumb questions.” In my teaching, this means making time for one-on-one conversations, using collective responses (such as “hum votes”) instead of cold-calling, and allowing anonymous questions on forums like Piazza. I have also helped **build tools that give students ways to work through confusion privately**. In Brown’s *Accelerated Introduction to Computer Science*, students can check their understanding of assignments by testing examples against a hidden reference implementation [1]. In *Logic for Systems*, they can receive scaffolded AI assistance within their IDE [2].

Questions for checking understanding. Once students feel safe being wrong, their answers become opportunities to understand their reasoning and adapt my teaching. For this to work, they need opportunities for formative feedback: ways to check their understanding without worrying that an incorrect answer will cost them a grade. When helping a student work through a problem, I ask what they expect to happen and why, then use their reasoning to decide whether to ask another question, offer a hint, work through an example, or explain directly. I’ve also found that **incorrect answers can reveal student misconceptions** [3]. I have used this insight to inform assignment design for both *Accelerated Introduction to Computer Science* and *Logic for Systems* courses at Brown. I’ve also applied this insight in developing the *LTL Tutor*², which has been **used by nearly 700 learners** over the past two years, both in formal methods courses at multiple universities and for self-directed study.

Questions as invitations to discovery. What excites me most about teaching through dialogue is how it can open avenues of inquiry anticipated by neither the student nor the teacher. For the past four years, I have helped run undergraduate TA training for Brown’s *Logic for Systems* and *Programming Languages* courses. Rather than reviewing sample solutions, I **put my research on student misconceptions into practice** through a pair-share activity centered on the question: *How might students get this assignment wrong?* We then come together as course staff to discuss possible misconceptions and collectively develop strategies for guiding students through them. I find this open-ended exercise productive at multiple levels. It not only helps TAs revisit familiar material through analysis rather than review, but also develops their ability to teach it. Their responses often surface possibilities I hadn’t anticipated, helping us improve our grading rubrics and test suites.

Teaching Experience & Mentoring

As an undergraduate TA for *Programming Languages* at Tufts University, I led weekly recitation sessions for three semesters. It was in these small-group sessions (~10 students each) that I discovered how much I loved teaching, particularly using questions to guide students through difficult concepts. I've sought to improve as a teacher ever since. When I began graduate school at Brown, I enrolled in the university's Sheridan Teaching Seminar to develop a stronger foundation in pedagogy. I've since put that training into practice as a TA for *Logic for Systems* and *Programming Languages*, through office hours, guest lectures, and training other TAs.

My teaching work has also extended to curriculum and tool development. I helped build Forge [4], a pedagogical **specification language** used in Brown's *Logic for Systems*, and helped design assignments for the course. I have also developed educational tools that instructors can use in their own courses (some discussed above). I'm particularly proud that my paper on the LTL Tutor [5] received a **Distinguished Paper Award at CAV**, recognition of an *educational* tool at a leading *technical* conference.

I have also been fortunate enough to **guide interns at Microsoft** and students at Brown, whose goals have ranged from gaining software engineering experience to exploring if research is for them. Through these experiences, I've learned that I enjoy a high-touch approach to mentoring, with regular meetings and structured time for self-reflection. I use these check-ins to discuss not just how the project is progressing, but whether the student is getting the experience they hoped for. These conversations often challenge my own assumptions about the work, reminding me that teaching through dialogue is as much about listening and learning as it is about guiding.

Teaching Interests

My background spans a broad range of software-related disciplines. My deepest expertise lies in **programming languages and formal methods**, and I am well equipped to teach courses in programming languages, compilers, and related topics. My time **working on large software systems like Windows and Azure** also equips me to teach courses in software engineering and systems.

For over half a decade, I have helped develop Brown's *Logic for Systems*, which uses real-world systems as an on-ramp to formal reasoning [6]. This approach reflects my research group's conviction that **technical rigor need not come at the expense of approachability**. I would be excited to teach a course along similar lines. Students would learn to model systems they encounter in other courses and in practice, reason about those models using logic, and understand the algorithms underlying automated reasoning tools. As generative AI makes it easier to churn out vast quantities of code, I believe every computer science curriculum will need a course that teaches students to reason about what that code actually does.

I am also excited for the opportunity to develop new courses. During my time in industry, I observed that **my most impactful coworkers were those who could effectively communicate what they needed from software** or what was wrong with it. As generative AI shifts the emphasis of programming from writing implementations to specifying intent, working effectively with software will increasingly require articulating what it should do, not just reasoning about what it does. I am keen to develop a course on *Specifications, Software, and AI* that draws on applied cognitive science to study the informal and semi-formal ways people articulate and communicate technical intent. Students would learn to specify intent at various levels, ranging from semi-structured prose (design docs, bug reports) to **checkable artifacts** (property-based tests, formal models). They would also learn to recognize what their specifications constrain, what they leave open, and how to balance ambiguity and precision based on the context.

I am also equipped to teach introductory programming courses. While I appreciate the variety of factors that make such courses demanding to teach, I believe that my computing education research has equipped me to meet these challenges. My philosophy is that these courses should be designed to empower students from a broad range of backgrounds, not weed them out.

References

- [1] John Wrenn and Shriram Krishnamurthi. “Executable Examples for Programming Problem Comprehension”. In: *Proceedings of the 2019 ACM Conference on International Computing Education Research*. ICER '19. Toronto ON, Canada: Association for Computing Machinery, 2019, pp. 131–139. ISBN: 9781450361859. DOI: [10.1145/3291279.3339416](https://doi.org/10.1145/3291279.3339416). URL: <https://doi.org/10.1145/3291279.3339416>.
- [2] Siddhartha Prasad, Ben Greenman, Tim Nelson, and Shriram Krishnamurthi. “Generating Programs Trivially: Student Use of Large Language Models”. In: *Proceedings of the ACM Conference on Global Computing Education Vol 1*. CompEd 2023. Hyderabad, India: Association for Computing Machinery, 2023, pp. 126–132. ISBN: 9798400700484. DOI: [10.1145/3576882.3617921](https://doi.org/10.1145/3576882.3617921). URL: <https://doi.org/10.1145/3576882.3617921>.
- [3] Siddhartha Prasad, Ben Greenman, Tim Nelson, John Wrenn, and Shriram Krishnamurthi. “Making Hay from Wheats: A Classsourcing Method to Identify Misconceptions”. In: *Proceedings of the 22nd Koli Calling International Conference on Computing Education Research*. Koli Calling '22. Koli, Finland: Association for Computing Machinery, 2022. ISBN: 9781450396165. DOI: [10.1145/3564721.3564726](https://doi.org/10.1145/3564721.3564726). URL: <https://doi.org/10.1145/3564721.3564726>.
- [4] Tim Nelson, Ben Greenman, Siddhartha Prasad, Tristan Dyer, Ethan Bove, Qianfan Chen, Charles Cutting, Thomas Del Vecchio, Sidney LeVine, Julianne Rudner, Ben Ryjikov, Alexander Varga, Andrew Wagner, Luke West, and Shriram Krishnamurthi. “Forge: A Tool and Language for Teaching Formal Methods”. In: *Proc. ACM Program. Lang.* 8.OOPSLA1 (Apr. 2024). DOI: [10.1145/3649833](https://doi.org/10.1145/3649833). URL: <https://doi.org/10.1145/3649833>.
- [5] Siddhartha Prasad, Ben Greenman, Tim Nelson, and Shriram Krishnamurthi. “A Misconception-Driven Adaptive Tutor for Linear Temporal Logic”. In: *CAV 2025*. 2025. DOI: [10.1007/978-3-031-98685-7_9](https://doi.org/10.1007/978-3-031-98685-7_9).
- [6] Shriram Krishnamurthi, Tim Nelson, et al. “Logic for Systems: A Gradual Introduction to Formal Methods”. In: *Bulletin of EATCS* 145.1 (2025).