Can reading make you better at programming? Do developers think cannabis leads to software creativity? What are student bugs like outside of the classroom? I am intrigued by the multifaceted ways software engineering research can help programmers be more productive and supported. My work directly ameliorates differences for under-supported groups through novel algorithms and theoretically-grounded interventions that help novices act like experts faster. By combining large-scale empirical analyses and controlled human-focused experimental design, I not only mathematically model how understudied features impact programmer productivity but also provide actionable and evidence-based interventions for diverse programmer groups.

I do so via three lenses: leveraging program synthesis and machine learning to *develop efficient and usable programming support*, using cognitive insights to *design effective programming training*, and *identifying and addressing non-technical productivity barriers*. I make use of creative and novel experimental design, large-scale evaluations (with millions of programming interactions or hundreds of human subjects), and interdisciplinary skill sets outside of core software engineering techniques (including programming languages, machine learning, psychology, and medicine).

My PhD research has contributed to 16 peer-reviewed publications, including several in top Software Engineering and Programming Languages venues (ICSE, ESEC/FSE, PLDI, OOPSLA, ASE). My research on increasing involvement in open source for social good received a distinguished paper award at ESEC/FSE in 2023 [11]. I have experience securing funding ($8,000 from the NSF, $74,700 from the University of Michigan), and I was awarded the NSF GRFP. I have acted as primary advisor to seven undergraduates, leading to four peer-reviewed publications (three with student first authors). I believe my experiences position me well to succeed as a tenure-track Assistant Professor of Computer Science at University X.

Developing Efficient and Usable Programming Tools: *Can we combine programming languages techniques and machine learning insights to support writing more correct code faster?*

I improve programmer productivity with efficient and usable techniques for fixing software defects, a task crucial for streamlining software production and ensuring reliability. My primary insight is to augment program repair approaches with emerging techniques from program synthesis and machine learning. For example, I helped design SEQ2PARSE, which uses a novel neurosymbolic approach to make error-correcting parsers efficient and fix parse errors [16]; RITE, which learns and synthesizes type-driven repairs [15]; and an approach using LLMs to formalize natural language intent into postconditions that catch real-world defects [4]. I believe interdisciplinary collaboration takes time and effort to effectively communicate formalisms [7]. My collaborations with Ranjit Jhala at UCSD and with Shuvendu Lahiri and Sarah Fakhoury at Microsoft have led to four peer-reviewed publications [2, 8, 15, 16], with a fifth under submission (see tech report [4]).

I want to support novices who are learning programming without the traditional classroom or who find debugging challenging. Free tutoring environments try to help, but suffer from low retention, reducing their reach in practice. Through large-scale empirical investigations of one such platform, Python Tutor, I found that non-traditional novices struggle with different errors than those faced by experts, errors not well supported by existing tools [8, 16]. Many of my techniques are designed with these novice-specific errors in mind [2, 8, 16]. For example, I created INFIX, a template-based repair approach that suggests a correct fix for 95% of input-related bugs, which I identified as particularly challenging for novices [8].

I use large-scale human-focused evaluations to assess the effectiveness and impact of software tools in practice, leveraging both historical datasets and human subjects. For example, using the

Python Tutor data, I compare my techniques to tens of thousands of historical student bug fixes. Gaining access to this dataset required significant communication with its maintainer, Philip Guo. However, I believe that this effort is worthwhile, as large-scale evaluations can help mitigate bias and improve the generalizability of research results. Additionally, I have conducted four human studies of programming tools, involving over 200 participants of varying expertise [2, 8, 15, 16]. I have found that my tools produce high quality repairs on par with human-made fixes. For example, 78% of SEQ2PARSE repairs met or exceeded the quality of the historical repair.

Designing Effective Developer Training: *Can we use objective measures of cognition to inform developer training and improve programming outcomes?*

I believe in going beyond targeted tools to mitigate productivity barriers more broadly for diverse groups. To provide support that generalizes to many software aspects, I am interested in designing effective programming training—but what skills should we train? My key insight is to use medical imaging to construct mathematical models relating observed programmer expertise to brain activation patterns. Based on these models, I design supplemental training that *transfers* to improved programming outcomes. For example, I used fNIRS to model novice programmer cognition [6]. I then designed an 11-week curriculum to train one identified skill, technical reading ability. In a controlled longitudinal study, my intervention helped programmers read and trace through code better than those in a strong active control [5]. I have used a similar blend of expertise models and validated interventions for other aspects of software, including data structure manipulation [1, 9], debugging information search [13], and successful open-source contributions [11]. I find this approach exciting because my validated interventions could broaden participation in computing for students with lower incoming preparation in relevant skills.

I also enjoy interdisciplinary collaboration between computing and psychology. I have found that, through fostering cross-departmental collaborations, focusing on experimental design, and securing sufficient experimental resources, my approach yields impactful results. I make sure to work closely with psychology collaborators to ensure that I am applying concepts correctly, leading to two papers with psychology co-authors [5, 6]. I enjoy working with these collaborators to design human studies that isolate cognitive processes and maximize statistical power, even in a noisy programming context. I have also started pre-registering my hypotheses (see [1, 12]). While less common in software, this practice is used in other fields to mitigate bias and facilitate replication. Finally, longitudinal neuroimaging studies require significant resources, and thus I have taken the initiative to secure funding by identifying opportunities, conducting audience analysis, and assembling the right proposal team. Through this approach, I have received over $60,000 for my transfer training work. This funding allowed me to recruit 100 participants for my semester-long longitudinal study, as well as to support five undergraduate research assistants.

Illuminating Understudied Societal Productivity Barriers: *Can we turn anecdote into evidence and improve societal productivity factors for computing?*

I am interested in evidence-based understandings of societal productivity factors because of their potential to inform policy that supports underserved groups. I thus augment my more-traditional research on tools and training with work on cultural or environmental factors that impact software. I first identify a societal factor with anecdotal connections to software, but limited evidence. I then conduct the research thoroughly and ethically, taking care to use the right methods.

Here, I focus on one factor, psychoactive substance use, as a microcosm of my approach. My interest was sparked when I observed a fellow researcher use cannabis to help with a programming proof. Surprised by this, I surveyed the literature for the scientific consensus about whether the effect generalized. Despite conflicting anecdotal claims on developer forums, I found virtually no empirical research. My curiosity piqued, I developed and executed on a research plan to turn anecdote into evidence. Given the complexity of societal interactions around substance use, I chose a mixed-methods approach. I first confirmed the relevance of psychoactive substances in software via a qualitative study [14] and a survey with over 800 programmers [3]. I then conducted an observational study of the impact of one substance, cannabis, on programming productivity, finding significant evidence of impairment [12]. I used rigorous experimental controls and pre-registered hypotheses to promote generalizability and increase the impact of my findings.

Care must be taken to thoroughly and ethically investigate such stigmatized external factors. I consulted my Institutional Review Board on best practices for balancing participant safety and experimental rigor. I have found that contacting the IRB in advance of the main protocol is helpful for conducting ethical human subjects research. My efforts have led to 11 successful protocols, including four on psychoactive substances. I also reached out to medical researchers, and my initiative led to both a successful research collaboration (with a co-authored paper [3]) and also to resources via an $8,000 grant from Michigan's NIH-funded Psychedelics Research Center.

I have found that societal considerations are approachable and interesting for student mentees: this lens into productivity has led to two papers with student first authors [12,14]. My research has also sparked considerable interest from developers, as evidenced by multiple public press articles and social media discussions. This demonstrates the relevance of bridging academic research with real-world issues. In essence, my approach not only reveals hidden aspects of programmer productivity but also transforms anecdotal evidence into practical, scientific knowledge, contributing to a more comprehensive understanding of effective software development.

Toward More Supported, Inclusive, and Productive Software Development

In the short term, I am considering projects for all three lenses. I plan to continue collaborating with Microsoft on improving AI-powered programmer support via controlled user evaluations in industrial contexts. Building on my work on cognition and programming, I plan to further explore the mechanisms behind how technical reading transfers to programming to develop even more efficient training curricula. I will also apply my approach for converting anecdotal observations into evidence to other societal factors impacting programming productivity. For topics, I will use insights from my previous research, personal experiences, and the interests of my students. This approach has already proven successful: I am collaborating with a former mentee (now a Carnegie Mellon PhD student), on a project supporting neurodiverse programmers. This topic is both personally significant to her and also motivated by previous interviews I've conducted with neurodiverse, substance-using developers who expressed a need for additional support.

In the long term, my research vision is *a paradigm shift both in who programs and also in prioritizing effective technical communication as a driving force in software*. As AI and low-code environments become more prevalent and programming becomes more accessible to those without formal training, I foresee a need to support and help a wider range of individuals communicate precise ideas. My focus will be on creating training and support mechanisms to help people effectively and efficiently read, understand, communicate, and verify technical formalisms and programs (be they human- or machine-written). In advancing a communication-first programming paradigm, I aim to leverage my research on technical reading and machine learning-enhanced programming tools, focusing on societal factors and human-centric software development. My achievements in research, funding, and mentorship highlight my readiness for a tenure-track faculty role, where I am excited to work for a future with more supported, inclusive, and productive programmers.

References

[1] Hammad Ahmad, Madeline Endres, Kaia Newman, Priscila Santiesteban, Emma Shedden, and Westley Weimer. Causal relationships and programming outcomes: A transcranial magnetic stimulation experiment. In *International Conference on Software Engineering*, 2024.

[2] Benjamin Cosman, Madeline Endres, Georgios Sakkas, Leon Medvinsky, Yao-Yuan Yang, Ranjit Jhala, Kamalika Chaudhuri, and Westley Weimer. Pablo: Helping novices debug python code through data-driven fault localization. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*, pages 1047–1053, 2020.

[3] Madeline Endres, Kevin Boehnke, and Westley Weimer. Hashing it out: A survey of programmers' cannabis usage, perception, and motivation. In *International Conference on Software Engineering*, page 1107–1119, 2022.

[4] Madeline Endres, Sarah Fakhoury, Saikat Chakraborty, and Shuvendu K Lahiri. Formalizing natural language intent into program specifications via large language models. *arXiv preprint arXiv:2310.01831*, 2023.

[5] Madeline Endres, Madison Fansher, Priti Shah, and Westley Weimer. To read or to rotate? comparing the effects of technical reading training and spatial skills training on novice programming ability. In *Foundations of Software Engineering*, pages 754–766, 2021.

[6] Madeline Endres, Zachary Karas, Xiaosu Hu, Ioulia Kovelman, and Westley Weimer. Relating reading, visualization, and coding for new programmers: A neuroimaging study. In *International Conference on Software Engineering*, pages 600–612, 2021.

[7] Madeline Endres, Pemma Reiter, Stephanie Forrest, and Westley Weimer. What can program repair learn from code review? In *Proceedings of the Third International Workshop on Automated Program Repair*, pages 33–34, 2022.

[8] Madeline Endres, Georgios Sakkas, Benjamin Cosman, Ranjit Jhala, and Westley Weimer. Infix: Automatically repairing novice program inputs. In *34th IEEE/ACM International Conference on Automated Software Engineering, ASE 2019, San Diego, CA, USA, November 11-15, 2019*, pages 399–410. IEEE, 2019.

[9] Madeline Endres, Westley Weimer, and Amir Kamil. An analysis of iterative and recursive problem performance. In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*, pages 321–327, 2021.

[10] Madeline Endres, Westley Weimer, and Amir Kamil. Making a gamble: Recruiting se participants on a budget. In *1st Workshop on Recruiting Participants for Empirical Software Engineering*, 2022.

[11] Zihan Fang, Madeline Endres, Thomas Zimmermann, Denae Ford, Westley Weimer, Kevin Leach, and Yu Huang. A four-year study of student contributions to oss vs. oss4sg with a lightweight intervention. In *Proceedings of the Symposium on the Foundations of Software Engineering*, 2023.

[12] Wenxin He, Manasvi Parikh, Westley Weimer, and Madeline Endres. High expectations: an observational study of programming and cannabis intoxication. In *Conditional Acceptance at International Conference on Software Engineering*, 2024.

[13] Annie Li, Madeline Endres, and Westley Weimer. Debugging with stack overflow: Web search behavior in novice and expert programmers. In *Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: Software Engineering Education and Training*, pages 69–81, 2022.

[14] Kaia Newman, Madeline Endres, Westley Weimer, and Brittany Johnson. From organizations to individuals: Psychoactive substance use by professional programmers. In *International Conference on Software Engineering*, pages 665–677, 2023.

[15] Georgios Sakkas, Madeline Endres, Benjamin Cosman, Westley Weimer, and Ranjit Jhala. Type error feedback via analytic program repair. In *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 16–30, 2020.

[16] Georgios Sakkas, Madeline Endres, Philip J Guo, Westley Weimer, and Ranjit Jhala. Seq2parse: neurosymbolic parse error repair. *Proceedings of the ACM on Programming Languages*, 6(OOPSLA2):1180–1206, 2022.

[17] Priscila Santiesteban, Madeline Endres, and Westley Weimer. An analysis of sex differences in computing teaching evaluations. In *Proceedings of the Third Workshop on Gender Equality, Diversity, and Inclusion in Software Engineering*, pages 84–87, 2022.