Three Lenses for Improving Programmer Productivity

From Anecdote to Evidence

Madeline Endres, PhD Defense, University of Michigan
Why study human-focused programming productivity?

### The Range of Individual Differences in Programming Performance

*Sackman (et al.), 1968*

<table>
<thead>
<tr>
<th>Performance Measure</th>
<th>Slowest Coder</th>
<th>Fastest Coder</th>
<th>Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code Hours: Algebra Problem</td>
<td>111</td>
<td>7</td>
<td>16:1</td>
</tr>
<tr>
<td>Code Hours: Maze Problem</td>
<td>50</td>
<td>2</td>
<td>25:1</td>
</tr>
<tr>
<td>Debug Hours: Algebra Problem</td>
<td>170</td>
<td>6</td>
<td>28:1</td>
</tr>
<tr>
<td>Debug Hours: Maze Problem</td>
<td>26</td>
<td>1</td>
<td>26:1</td>
</tr>
</tbody>
</table>
Why study human-focused programming productivity?

The Range of Individual Differences in Programming Performance  
*Sackman (et al.), 1968*

<table>
<thead>
<tr>
<th>Performance Measure</th>
<th>Slowest Coder</th>
<th>Fastest Coder</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code Hours: Algebra Problem</td>
<td>111</td>
<td>7</td>
</tr>
<tr>
<td>Code Hours: Maze Problem</td>
<td>50</td>
<td>2</td>
</tr>
<tr>
<td>Debug Hours: Algebra Problem</td>
<td>170</td>
<td>6</td>
</tr>
<tr>
<td>Debug Hours: Maze Problem</td>
<td>26</td>
<td>1</td>
</tr>
</tbody>
</table>

**Novice Software Developers, All Over Again**
Andrew Begel & Beth Simon

**A Tale of Two Cities: Software Developers Working from Home during the COVID-19 Pandemic**
DENAE FORD, Microsoft Research

**Socioeconomic Status and Computer Science Achievement**
Miranda C. Parker, Amber Solomon, and Brianna Pritchett

**A Large-Scale Survey on the Usability of AI Programming Assistants: Successes and Challenges**
Jenny T. Liang, Chenyang Yang, and Brad A. Myers

**What Predicts Software Developers’ Productivity?**
Emerson Murphy-Hill, Ciera Jaspan, Caitlin Sadowski, David Shepherd, Michael Phillips, Collin Winter, Andrea Knight, Edward Smith, and Matthew Jorde
Developing Efficient and Usable Programming Support

Can we support non-traditional novices in writing more correct code faster?

Designing Effective Developer Training

Can we use cognitive insights to inform training and improve programming outcomes?

Understanding External Productivity Factors

How does psychoactive substance use impact software productivity?
## Improving Programming Productivity: My Human-Focused Approach

<table>
<thead>
<tr>
<th>Desired Research Attribute</th>
<th>Why I'm Excited (and you could be too!)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Provide <em>Theoretically-Grounded</em> and <em>Actionable Insights</em></td>
<td>Bridging the gap between novel theoretical ideas to supporting programmers in practice leads to higher impact</td>
</tr>
</tbody>
</table>
## Improving Programming Productivity: My Human-Focused Approach

<table>
<thead>
<tr>
<th>Desired Research Attribute</th>
<th>Why I'm Excited (and you could be too!)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Provide Theoretically-Grounded and Actionable Insights</strong></td>
<td>Bridging the gap between novel theoretical ideas to supporting programmers in practice leads to higher impact</td>
</tr>
<tr>
<td><strong>Include Empirical or Objective Measures of Programmers</strong></td>
<td>Captures aspects of programming beyond self-reporting alone, including unconscious behaviors and habits</td>
</tr>
</tbody>
</table>
### Improving Programming Productivity: My Human-Focused Approach

<table>
<thead>
<tr>
<th>Desired Research Attribute</th>
<th>Why I'm Excited (and you could be too!)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Provide <em>Theoretically-Grounded and Actionable Insights</em></td>
<td>Bridging the gap between novel theoretical ideas to supporting programmers in practice leads to higher impact</td>
</tr>
<tr>
<td>Include <em>Empirical or Objective Measures</em> of Programmers</td>
<td>Captures aspects of programming beyond self-reporting alone, including unconscious behaviors and habits</td>
</tr>
<tr>
<td><em>Minimize Scientific Bias</em> to Support Generalizability</td>
<td>Controlled experimental design can capture a signal, even for complex human behavior</td>
</tr>
<tr>
<td>Desired Research Attribute</td>
<td>Why I'm Excited (and you could be too!)</td>
</tr>
<tr>
<td>----------------------------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Provide <em>Theoretically-Grounded</em> and <em>Actionable Insights</em></td>
<td>Bridging the gap between novel theoretical ideas to supporting programmers in practice leads to higher impact</td>
</tr>
<tr>
<td>Include <em>Empirical or Objective Measures</em> of Programmers</td>
<td>Captures aspects of programming beyond self-reporting alone, including unconscious behaviors and habits</td>
</tr>
<tr>
<td><em>Minimize Scientific Bias</em> to Support Generalizability</td>
<td>Controlled experimental design can capture a signal, even for complex human behavior</td>
</tr>
<tr>
<td>Support <em>Diverse Developer Groups</em></td>
<td>I prefer approaches that not only help programmers in general, but also help those who need the most support</td>
</tr>
</tbody>
</table>
InFix and Seq2Parse: Developing Efficient and Usable Tools

Supporting Non-traditional Programming Novices via a two novel forms of bug-fixing support
Many People Want to Learn to Code

**Without** traditional classroom support

The online Python Tutor interpreter currently has 60,000 users per month

How do Codecademy's 45 million users learn to code?

- **Full-time courses**: only 1/3 took a full-time course
- **Online courses**: 35% said online courses were their primary method for learning

Coding bootcamps see huge enrollment increase
One Such Platform: **Python Tutor**

Python Tutor is a free online interpreter. It helps novices visualize arbitrary code execution.

Users are primarily **Novice Programmers**

Started in 2010, it has had over **150 million** users from **180 countries**

```python
def listSum(numbers):
    if not numbers:
        return 0
    else:
        (f, rest) = numbers
        return f + listSum(rest)

myList = (1, (2, (3, None)))
total = listSum(myList)
```
Parse Errors

- Syntax errors are, by far, the most common Python error type experienced by novice programmers (77%)

```python
u = 42
x = 3.14
print(x * math.e / 2
```

Input-Related Bugs

- We found that 6% of student errors are resolved by fixing the program input, not the source code

**Example Code and Input**

```python
u = 42
x = float(input())
print(x * math.e / 2
```

```
ValueError: could not convert string to float: '26,2'
```

```python
u = 42
x = 3.14
print(x * math.e / 2
```
Parse Errors

- Syntax errors are, by far, the most common Python error type experienced by novice programmers (77%)

Proposed Approach: Neurosymbolic technique, **Seq2Parse**

*Published in OOPSLA, 2022*

Input-Related Bugs

- We found that 6% of student errors are resolved by fixing the program input, not the source code

Example Code and Input

```
u = 42
x = float(input())
print(x * math.e / 2)
```

ValueError: could not convert string to float: '26,2'

```
Proposed Approach: Template-repair approach, **InFix**

*Published in ASE, 2019*
```
Parse Errors

- Syntax errors are, by far, the most common Python error type experienced by novice programmers (77%)

Proposed Approach: Neurosymbolic technique, **Seq2Parse**  
*Published in OOPSLA, 2022*

Input-Related Bugs

- We found that 6% of student errors are resolved by fixing the program input, not the source code

**Example Code and Input**

```python
u = 42
x = float(input())
print(x * math.e / 2)
```

ValueError: could not convert string to float: '26,2'

SyntaxError: missing parentheses in call to print

```python
u = 42
x = 3.14
print(x * math.e / 2)
```

Proposed Approach: Template-repair approach, **InFix**  
*Published in ASE, 2019*
What do Non-Traditional Novices Struggle with? *Parse Errors*

For Non-Traditional Novices, Parse Errors (Syntax Errors) are both **common** and **challenging**.

37% of Parse Errors take over two minutes to resolve.

More complex fixes take even longer.
Fixing Parse Errors: How can we support Novices?

Goal: We want support for fixing parse errors faced by non-traditional novices that is both:

● **Effective**: can provide **helpful repairs close to the user's intent** in the majority of cases

● **Efficient**: Fast enough to be computed in **real time**
**Fixing Parse Errors**: How can we support Novices?

**Goal**: We want support for fixing parse errors faced by non-traditional novices that is both:

- **Effective**: can provide **helpful repairs close to the user's intent** in the majority of cases
- **Efficient**: Fast enough to be computed in **real time**
Parsing Overview

Program $P$

```python
def foo(a):
    return a + 42

def bar(a):
    b = foo(a) + 17
    return b +
```

Grammar $G$

```
S  →  Stmts end_marker
Stmt → FuncDef | ExprStmt
      | RetStmt | PassStmt | ...
FuncDef → def name Params : Block
Block → \n indent Stmts dedent
```
Finding Parse Errors: Fault Localization

Program $P$

```python
def foo(a):
    return a + 42

def bar(a):
    b = foo(a) + 17
    return b +
```

Grammar $G$

```
S → Stmts end_marker
Stmts → Stmt \n | Stmt \n Stmts
Stmt → FuncDef | ExprStmt
      | RetStmt | PassStmt | ...
FuncDef → def name Params : Block
Block → \n indent Stmts dedent
```
Fixing Parse Errors: Error Correcting Earley Parsers

Program $P$

```
def foo(a):
    return a + 42

def bar(a):
    b = foo(a) + 17
    return b +
```

Grammar $G'$

```
S  →  Stmts end_marker
Stmts → Stmt \n | Stmt \n Stmts
Stmt → FuncDef | ExprStmt
     | RetStmt | PassStmt | ...
FuncDef → def name Params : Block
Block → \n indent Stmts dedent
New_S → S | S Insert
RetStmt → | E_return | E_return Args
E_return → return | $\epsilon$ | Replace
           | Insert return
E_number → number | $\epsilon$ | Replace
           | Insert number
...
```

21
Fixing Parse Errors: Error Correcting Earley Parsers

Program $P$

```python
def foo(a):
    return a + 42

def bar(a):
```

Too many rules!

Grammar $G'$

```
S → Stmts end_marker
Stmts → Stmt \n | Stmt \n Stmt
Stmt → FuncDef | ExprStmt
    | RetStmt | PassStmt | ...
FuncDef → def name Params : Block
Block → \n indent Stmts dedent
New_S → S | S Insert
RetStmt → | E_return | E_return Args
E_return → return | $\epsilon$ | Replace
    | Insert return
E_number → number | $\epsilon$ | Replace
    | Insert number
```
Fixing Parse Errors: Neural Approaches

Pros:

- Sequence classifiers can be **good at predicting edits** or repairs **similar to human behavior**
- Once trained, neural approaches **can be efficient**

Cons:

- Generally, **no guarantees** that the response will correct (e.g., actually parse), let alone be a minimal repair
- Neural approaches can be **confused by program context** not directly related to the parse error

```python
def foo(a):
    return a + 42

def bar(a):
    b = foo(a) + 17
    return b +
```
**SEQ2PARSE: Key Insight**

- EC-Parsers guarantee a correct fix, but are slow because they consider too many production rules, the vast majority of which are not needed to fix any given error.
- In contrast, Neural approaches are fast and leverage user patterns, but can be inaccurate or untrustworthy if used alone.

We propose to get the best of both worlds and efficiently and accurately suggest repairs in a neurosymbolic fashion:

1. Train sequence classifiers to predict the relevant EC-rules for a given program, instead of the next token or the full fix.
2. Use the predicted rules to synthesize a Parse Error repair via EC-Parsing.
**SEQ2PARSE**: Efficient Fixes for Novice Parse Errors

- Program With Parse Error
- Error-Correcting Earley Parser
- Fixed Program
**Seq2Parse**: Efficient Fixes for Novice Parse Errors

- **Program With Parse Error**
- **Relevant Error Rule Predictor (Sequence Classifier)**
- **Error-Correcting Earley Parser**
- **Fixed Program**

**Python Tutor Dataset**

- **Fixed Programs**
- **Parse Error Programs**
**SEQ2PARSE**: Efficient Fixes for Novice Parse Errors

- **Program With Parse Error**
- **Relevant Error Rule Predictor (Sequence Classifier)**
- **Error-Correcting Earley Parser**
- **Fixed Program**

How do we learn relevant error rules?

**Python Tutor Dataset**

- **Fixed Programs**
- **Parse Error Programs**
Additional Considerations for Learning EC-Production Rules

Ill-parsed Program Representation for Learning:

- **Problem**: Predicting relevant production rules using full buggy programs causes the model to be confused by irrelevant program context.
- **Our Solution**: Instead of standard token strings, develop semantics for Abstracted Token Sequences that concentrate information relevant to a given parse error and remove confusing context.

Mitigating Representational Ambiguity:

- **Problem**: While needed, this abstraction adds ambiguity into what parse tree should result from any given abstracted token sequence.
- **Our Solution**: Use fixed Python Tutor programs to learn a Probabilistic Context Free Grammar and resolve parsing ambiguities.
**Seq2Parse: Python Implementation**

- **Dataset:** Over **One Million Buggy/Fixed Program Pairs** from Python Tutor
  - Average abstracted token sequence is 43 tokens long
  - 15,000 random programs used for evaluation, the rest for model training

- **Error Rule Prediction Model Structure:**
  - **Transformer classifier** with six blocks, each with a fully-connected hidden layer of 256 neurons and 12 attention heads, **connected to a DNN-based classifier** with two fully-connected hidden layers.
  - Trained using an Adam optimizer, a variant of stochastic gradient descent for 50 epochs.

- **Model Output:** Top 20 most likely error production rules for a given Buggy Program
  - These rules are then fed into the Error Correcting Earley Parser
**Seq2Parse: Does it work? Yes!**

*Seq2Parse can fix most parse errors for non-traditional novices, in real time*

*and with the same, or better, quality to the novices themselves!*

**Repair Rate:** *Seq2Parse* can parse and repair up to 94.25% of programs with syntax errors.

**Efficiency:** *Seq2Parse* can parse and repair the vast majority of the test set in under 20 seconds in a median time of 2.1 seconds.

**Quality:** *Seq2Parse* generates the exact fix as the historical user up to 35% of the time! Of the remainder, *Seq2Parse* repairs are equivalent to or more useful than historical repairs 52% and 15% of the time, respectively.
**Seq2Parse:** Does it work? Yes!

We assess repair quality via a study with 39 programmers

Captured 527 subjective quality ratings for a corpus of 50 Seq2Parse/historical fix pairs

Compared the two pairs using standard statistical tests

![Buggy Python Program](image)

```
1    shift = [(9, 2, 4), (7, 6, 9)]
2    for i in shift:
3       for item in i:
4       print(i[1])
```

![Debugging Hint: Possible Fix](image)

```
1    shift = [(9, 2, 4), (7, 6, 9)]
2    for i in shift:
3       for item in i:
4       print(i[1])
```

**Python Error Message**

File "program.py", line 4
print(i[1])
^  
IndentationError: expected an indented block after 'for' statement on line 3

**Questions To Answer:**

1. Between 1 (not helpful) and 5 (very helpful), how helpful is the Python Error Message for debugging the program?

2. Between 1 (not helpful) and 5 (very helpful), how helpful is the provided Possible Fix for debugging the program?
Seq2Parse: Does it work? Yes!

**Repair Rate:** Seq2Parse can parse and repair up to 94.25% of programs with syntax errors.

**Efficiency:** Seq2Parse can parse and repair the vast majority of the test set in under 20 seconds in a median time of 2.1 seconds.

**Quality:** Seq2Parse generates the exact fix as the historical user up to 35% of the time! Of the remainder, Seq2Parse repairs are equivalent to or more useful than historical repairs 52% and 15% of the time, respectively.

*Seq2Parse can fix most parse errors for non-traditional novices, in real time and with the same, or better, quality to the novices themselves!*

32
Lens 1 — Summary: Developing Better Bug Fixing Support

- We identified parse errors and input-related bugs as a significant barrier for non-traditional novices in practice.
- We propose Seq2Parse, a neurosymbolic approach to fixing parse errors, and InFix, a template-based approach for fixing input-related bugs.
- Our preliminary results show that both tools produce repairs that are accurate, efficient, and of high quality, as judged by humans.
Developing Efficient and Usable Programming Support

Can we support non-traditional novices in writing more correct code faster?

Designing Effective Developer Training

Can we use cognitive insights to inform training and improve programming outcomes?

Understanding External Productivity Factors

How does psychoactive substance use impact software productivity?
TO READ OR TO ROTATE?

An example of how cognitive insights can inform effective programming interventions

ESEC/FSE, 2021, ICSE 2021
Novice programmers often struggle, especially those students with weaker preparatory education. This struggle may result from insufficient preparation in cognitive skills necessary for programming.
How can we help students?

Cognitive interventions (the supplemental training of a necessary cognitive skill) can help underprepared students succeed in many fields.
How can we help students?

Cognitive interventions (the supplemental training of a necessary cognitive skill) can help underprepared students succeed in many fields.

A writing-intensive course improves biology undergraduates’ perception and confidence of their abilities to read scientific literature and communicate science.

Sara E. Brownell,1 Jordan V. Price,2 and Lawrence Steinman2,3
How can we help students?

**Cognitive interventions** (the supplemental training of a necessary cognitive skill) can **help underprepared students succeed in many fields**

A writing-intensive course improves biology undergraduates’ perception and confidence of their abilities to read scientific literature and communicate science

Sara E. Brownell,¹ Jordan V. Price,² and Lawrence F. Klopfer³

---

**THE EFFECTS OF ORIGAMI LESSONS ON STUDENTS’ SPATIAL VISUALIZATION SKILLS AND ACHIEVEMENT LEVELS IN A SEVENTH-GRADE MATHEMATICS CLASSROOM**

---

A Qualitative Inquiry into the Effects of Visualization on High School Chemistry Students’ Learning Process of Molecular Structure

Susan Deratzou
How can we help students?

**Cognitive interventions** (the supplemental training of a necessary cognitive skill) can help underprepared students succeed in many fields.

A writing-intensive course improves biology undergraduates’ perception and confidence of their abilities to read scientific literature and communicate science.

Sara E. Brownell,^1^ Jordan V. Price,^2^ and Lawrence M. Henry^3^,^4^  

---

**A Qualitative Inquiry into the Effects of Visualization on High School Chemistry Students’ Learning**

---

**Does spatial skills instruction improve STEM outcomes? The answer is ‘yes’**

Sheryl Sorby^a,^*,^ Norma Veurink^b^, Scott Streiner^c^
Cognitive interventions may also help improve programming ability for novices...
Cognitive interventions may also help improve programming ability for novices...

... but what cognitive skills should we target?
Neuroimaging and Software Engineering

- Understanding the **cognitive basis of software engineering** is important.

- Neuroimaging allows us to **objectively measure** this cognitive basis by **directly observing brain activation** patterns while programming! (as opposed to self-reported data)

- Potential impact areas of neuroimaging include pedagogy, technology transfer, expertise, adult retraining.
What do we know up to 2023?

- Neuroimaging uses **contrast-based experiments** to compare **programming** activities to **other cognitive tasks**

<table>
<thead>
<tr>
<th>Neuroimaging Experiment</th>
<th>Is programming like Reading?</th>
<th>Is programming like Spatial Reasoning?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Siegmund et al., (2014)</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>Siegmund et al., (2017)</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>Floyd et al., (2017)</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>Huang et al., (2019)</td>
<td></td>
<td>✔</td>
</tr>
</tbody>
</table>
What do we know up to 2023?

- Neuroimaging uses **contrast-based experiments** to compare **programming** activities to **other cognitive tasks**

<table>
<thead>
<tr>
<th>Neuroimaging Experiment</th>
<th>Is programming like Reading?</th>
<th>Is programming like Spatial Reasoning?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Siegmund <em>et al.</em>, (2014)</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>Siegmund <em>et al.</em>, (2017)</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>**Floyd et al.*, (2017)</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>Huang <em>et al.</em>, (2019)</td>
<td></td>
<td>✔</td>
</tr>
</tbody>
</table>

Found connection with expertise
Neuroimaging uses contrast-based experiments to compare programming activities to other cognitive tasks.

<table>
<thead>
<tr>
<th>Neuroimaging Experiment</th>
<th>Is programming like Reading?</th>
<th>Is programming like Spatial Reasoning?</th>
<th>What about with novices?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Siegmund et al., (2014)</td>
<td>✓</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>Siegmund et al., (2017)</td>
<td>✓</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>Floyd et al., (2017)</td>
<td>✓</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>Huang et al., (2019)</td>
<td>✓</td>
<td>✓</td>
<td>?</td>
</tr>
</tbody>
</table>
Lens 2 Study Overview

Phase 1: **Neuroimaging**

- We first build a model of novice programmer cognition using the first neuroimaging study of true novice programmers during code comprehension

  *Published in ICSE, 2021*

Phase 2: **Transfer Training**

- We then investigate the usefulness of transfer training in computing comparing the impact of two cognitive interventions on novice programming performance in a controlled, longitudinal study

  *Published in FSE, 2021*
Phase 1: Neuroimaging Method

- We use **Functional Near Infrared Spectroscopy** (fNIRS) to capture the brain activation patterns of **novice programmers** (no prior programming experience)
  - fNIRS uses light to measure the oxygen levels in different parts of the brain
  - Supports studying the brain while doing natural programming tasks

- We compare programming-associated activations to **two well-understood cognitive tasks** commonly used in neuroimaging studies of expert developers: **spatial visualization** and **reading**
Experimental Timeline: A Semester of CS1

- **Week 1:** Start of the CS1 (EECS 183) semester
- **Week 3:** Participant recruitment from CS1
- **Week 4-5.5:** Brain scans
- **Week 16:** End of semester
Experimental Timeline: A Semester of CS1

Week 1: Start of the CS1 (EECS 183) semester

Week 3: Participant recruitment from CS1

Week 4-5.5: Brain scans

Week 16: End of semester
Neuroimaging Stimuli

We compare brain activation during three tasks:
Neuroimaging Stimuli

We compare brain activation during three tasks:

- CS1-Level Programming
Neuroimaging Stimuli

We compare brain activation during three tasks:

- CS1-Level Programming
- Mental Rotation
Neuroimaging Stimuli

We compare brain activation during three tasks:

- CS1-Level Programming
- Mental Rotation
- Prose Fill in the Blank
fNIRS Scan Data Collection and Analysis

- Each scan session lasts **two hours** in a darkened room
  - 90 stimuli, 30 of each type (programming, mental rotation, reading)

- 36 participants, **31 valid** (24 female, 7 male)
  - Recruited from EECS 183, here at Michigan!

- Data Analysis
  - Compare activation by task by brain area using best practices from psychology
  - Significance threshold: \( p < 0.01 \).
  - We used False Discovery Rate to correct for multiple comparisons: \( q < 0.05 \).
A Mathematical Model of Novice Cognition: Primary Research Questions

● **Comparative Activation**: How does novice program comprehension **compare to prose comprehension and spatial reasoning** at the cognitive level?
  ○ How do novices' brain activation patterns **compare to those of expert developers**?
Phase 1 Results: Comparative Brain Activation

- **Question:** Do novices use spatial and/or language areas while programming?

- **Result:** While areas associated with both are activated, we find more substantial differences between Coding and Reading than between Coding and Mental Rotation.
Phase 1 Results: Comparative Brain Activation

- **Question:** Do novices use spatial and/or language areas while programming?

- **Result:** While areas associated with both are activated, we find **more substantial differences between Coding and Reading** than between **Coding and Mental Rotation**.
Question: Do novices use spatial and/or language areas while programming?

Result: While areas associated with both are activated, we find more substantial differences between Coding and Reading than between Coding and Mental Rotation.
Phase 1 Results: Comparative Brain Activation

- Question: Do novices use spatial and/or language areas while programming?

- Result: While areas associated with both are activated, we find more substantial differences between Coding and Reading than between Coding and Mental Rotation.
Phase 1 Results: Comparative Brain Activation

- **Question:** Do novices use spatial and/or language areas while programming?

- **Result:** While areas associated with both are activated, we find more substantial differences between Coding and Reading than between Coding and Mental Rotation.
Phase 1 Results: Comparative Brain Activation

- Question: **Do novices use spatial and/or language areas while programming?**

- Result: While areas associated with both are activated, we find more substantial differences between Coding and Reading than between Coding and Rotation.
Phase 1 Results: Comparative Brain Activation

- **Question:** Do novices use spatial and/or language areas while programming?

- **Result:** While areas associated with both are activated, we find more substantial differences between Coding and Reading than between Coding and Rotation.

- We also find that for novices coding engages more working memory and is more cognitively challenging than does either mental rotation or prose reading.

So for novices, we see more differences with reading that spatial ability. Now what?
Phase 1 Results: Comparing to Experts

- Question: **How does this finding compare to previous studies with experts?**

- Floyd *et al.* found that coding and prose tasks are more similar in terms of neural activity for senior undergraduate than for mid-level undergraduates.

- Our results: **the pattern may continue to novices.** For less experienced programmers, **programming and reading show little cognitive similarity**.

- Implications for developer training and pedagogy:
  - Perhaps **spatial skills enable general problem solving** for novices, but **domain-specific programming strategies** use more **reading-associated** cognitive processes.
  - Directly training reading-based domain-specific strategies may help **novices become experts faster**.
For novices, spatial reasoning is less distinct to programming than reading at a cognitive level.

This is in contrast to results with expert developers, and has implications for future programming training or interventions.
Phase 2: Transfer Training

A Tale of Two Cognitive Interventions

Standardized and Validated Spatial Reasoning Training

vs.

Our Novel CS-focused Technical Reading Training
Intervention 1: Spatial Reasoning Training

- **Spatial Reasoning** is the ability to mentally manipulate 2D and 3D shapes

- We use a **validated pre-made Spatial Reasoning Training Curriculum** developed for engineering students
  - Developed by Sorby *et al.* (2000)

- Includes sketching practice of shape rotation projection, and folding
Intervention 2: Technical Reading Training

- We developed an intervention to teach **strategies** for efficiently understanding scientific writing.

- Strategies focused on using **structural cues to scan academic papers** to retrieve and understand key points:
  - Inspired by eye tracking findings: experienced programmers tend to read **code non-linearly**, focusing on high level features.
**Objective measures of cognition**

Identify Relevant Cognitive Skills

- **Spatial Visualization**
- **Technical Reading**

---

**Experts Reading Code**

```java
import java
import as
import FixFactory as theory_factory

class Fix:
  def __init__(self, index, config, index_log):
    self.config = index.config
    self.log = index_log

def find_tutorial(self, file_name):
  ...  # This function finds all the tutorials by the initializing configuration
  filter is an optional filter that says which sessions to consider
  ...

  # see if the config file is indicating that this run is resuming a previous partial run
  to_return = 1

  # load the partial file if there is one
  if self.config.partial_file is not None:
    with open(f'{self.config.partial_file}_{self.r}'): f:
      f = self
      to_return = 1

  for line in f:
    try:
      if line.startswith('NEXT: '):
        ...  # make sure we have a valid config object
        general_config = json.loads(line)
        if len(experiment_results) == 0:
          print('This experiment was not actually completed')
        else:
          to_return.append((general_config, experiment_results))
    except Exception:
      pass

  # make dictionary of values
  zzz = set()
  for config, result in to_return:
    zzz.add(config['unique_id'])

  theorems = self.config.theorems()
  global_theory_config = self.config.get_specified_theoryⲔnbsp;for i in theorems
  global_theory_config = theory.factory.get_theory(zzz, global_theory_config, self.log)

  # new, loop through all the files in path
  base_path = self.config.get_session_path()
  count = 1
  files = []
  for folder in os.listdir(base_path):
    files = [f'{folder}/']
    print(f'Number: {i}_{folder}
print(folder)
  # check here if the folder has already been done by the thing in the config
  if folder in zzz:
```
Objective measures of cognition

Identify Relevant Cognitive Skills

Spatial Visualization
Technical Reading

Novices Reading Code

Busjahn, et al., 2015

Experts Reading Code
Semester CS1 Course With Final Exam
Spatial Training

Reading Training
Transfer Training Results: Which Group Did Better?

Spatial Reasoning Training

Technical Reading Training
Transfer Training Results: Which Group Did Better?

Spatial Reasoning Training

Technical Reading Training
Transfer Training Results: Which Group Did Better?
Now that we know that our Reading Training transferred to CS1, what programming skill did it help?
Now that we know that our Reading Training transferred to CS1, what programming skill did it help?

Our final programming assessment (the SCS1) had three types of questions: code completion, definitional, and code tracing.
How did the Reading Training Help?

Code Completion Questions

Definitional Questions

Tracing Questions

Reading

Spatial
How did the Reading Training Help?

$p = 0.03$
We compared the effects of **Spatial Reasoning Training** and our novel **CS-focused Technical Reading Training** on CS1 students.

We found that our **Technical Reading Training helped programming ability more**, especially **helping novices trace through code**.
Transfer Training Results: Summary

We compared the effects of Spatial Reasoning Training and our novel CS-focused Technical Reading Training on CS1 students.

We found that our Technical Reading Training helped programming ability more, especially helping novices trace through code.
Developing Efficient and Usable Programming Support

Can we support non-traditional novices in writing more correct code faster?

Designing Effective Developer Training

Can we use cognitive insights to inform training and improve programming outcomes?

Understanding External Productivity Factors

How does psychoactive substance use impact software productivity?
Psychoactive Substances and Programming?

A case study on how understudied external factors can impact software productivity

ICSE 2022, 2024

Credit: XKCD Comic, https://xkcd.com/323/
CULTURE OF PSYCHOACTIVE SUBSTANCE USE AND SOFTWARE

Based upon my experiences and observations:

- caffeine
- nicotine
- alcohol
- ritalin
- modafinil

I've never met a developer that didn’t use one of the aforementioned drugs during work.

Coder’s High

Programming is just like drugs, except the dealer pays you.

BY DAVID AUERBACH  JUNE 17, 2014 • 12:02 PM

“Taking LSD was a profound experience, one of the most important things in my life”

- Steve Jobs

Under pressure, Silicon Valley workers turn to LSD microdosing
However, this culture may conflict with some organizational structures – Take cannabis-related policies as an example:

We have a strict drug and alcohol policy. Employees are not permitted to use, possess, sell, transfer, manufacture, distribute, or be under the influence of illegal drugs on Cisco-owned or leased property, during working hours, while on company business, or while using company property.

Although certain jurisdictions may allow the prescription or other use of marijuana, this policy also applies to marijuana, which remains illegal under U.S. Federal law. Employees are not permitted to use, possess, sell, transfer, manufacture, distribute or be under the influence of these drugs while on Cisco owned or leased property, during working hours, while on company business, or while using company property. In addition to discipline for violation of this policy, the employee may be required to seek treatment to help prevent future violations.

29% of software developers have taken a drug test for a programming-related job. (Endres et al, 2022)

The FBI Says It Can't Find Hackers to Hire Because They All Smoke Pot

The FBI is struggling to find good hackers because of marijuana rules
Despite this conflict, little empirical research has been conducted on psychoactive substance use in software development.

We want to know if, when, or why developers use substances while programming: tech companies cannot effectively evaluate existing anti-drug policies.

And we want to build a mathematical model of how such substances actually impact programmers.

We present results from the:

First large-scale empirical study of psychoactive substance use in software engineering
(800+ programmers, 450 full-time devs)

First controlled observational study of cannabis and programming
(70+ programmers, pre-registered hypotheses)
Psychoactive Substances Exploratory Survey: **Summary**

803 survey responses:
- 440 from GitHub Emails
- 339 from University of Michigan
- 24 from Social Media
- 56% Have full-time programming jobs, 36% are students

---

**Madeline Endres**

Hello! Do you program or are you in a programming-related field? If so,
Psychoactive Substances Exploratory Survey: **Summary**

**803 survey responses:**
- 440 from GitHub Emails
- 339 from University of Michigan
- 24 from Social Media
- 56% Have full-time programming jobs, 36% are students

**Usage While Programming in Last Year**

- Alcohol: 24.53%
- Cannabis: 24.40%
- Tobacco: 5.73%
- Amphetamines: 4.73%
- Hallucinogens: 2.12%

---

**Madeline Endres**
Hello! Do you program or are you in a programming-related field? If so,
Psychoactive Substances Exploratory Survey: **Summary**

### 803 survey responses:
- 440 from GitHub Emails
- 339 from University of Michigan
- 24 from Social Media
- 56% Have full-time programming jobs, 36% are students

### Usage While Programming in Last Year

- **Alcohol**: 24.53%
- **Cannabis**: 24.40%
- **Tobacco**: 6.73%
- **Amphetamines**: 4.73%
- **Hallucinogens**: 2.12%

- 33% use for *work-related tasks*
- 11% use at a *frequency likely to be caught by a drug test*
A Controlled Observational Study: **Cannabis**

- **Goal:** To **build a mathematical model of how cannabis use impacts programming.**
  - We want our model to be rigorous enough to be used by individual developers and policy makers alike in making more informed cannabis and programming decisions.
  - We **pre-registered our hypotheses** to facilitate future replication.
A Controlled Observational Study: Cannabis

- Goal: To build a mathematical model of how cannabis use impacts programming.
  - We want our model to be rigorous enough to be used by individual developers and policy makers alike in making more informed cannabis and programming decisions.
  - We pre-registered our hypotheses to facilitate future replication.

- Design Considerations:
  - Achieving sufficient statistical power to answer our pre-registered research questions
  - Balancing ecological validity with experimental control
  - Maximizing participant privacy and safety
Remote Programming
Session 1

Programming While High

Programming While Sober

```python
class Solution:
    """"""" You are given a string 's' consisting of lowercase English letters. A duplicate removal consists of choosing two adjacent and equal letters and removing them. We repeatedly make duplicate removals on 's' until we no longer can. Return the final string after all such duplicate removals have been made. It can be proven that the answer is unique. Full stimulus has 10 Examples and input constraints here """"
    def removeDuplicates(self, s: str) -> str:
        return """"

    class Test(object):
        def test_removeDuplicates(self):
            print("""Test 1"""")
            solution = Solution()
            answer1 = solution.removeDuplicates("abbaca")
```

```bash
> /workspaces/CodeSpaceTest (main) $
```

---

96
Programming While High

Programming While Sober

Remote Programming Session 1

Remote Programming Session 2

```
# Problem

class Solution:
    def removeDuplicates(self, s: str) -> str:
        stack = []
        for c in s:
            if stack and stack[-1] == c:
                stack.pop()
            else:
                stack.append(c)
        return ''.join(stack)
```
Programming While High

Programming While Sober

Remote Programming Session 1

Remote Programming Session 2

?
Results: Pre-registered Hypotheses

**RQ1:** How does cannabis intoxication while programming impact program correctness?

- Hypothesis: Programs will be **less correct** when written by cannabis-intoxicated programmers.

**RQ2:** How does cannabis intoxication while programming impact programming speed?

- Hypothesis: Cannabis-intoxicated programmers **will take longer to write** programs.
Results: Pre-registered Hypotheses

**RQ1:** How does cannabis intoxication while programming impact program correctness?

- Hypothesis: Programs will be **less correct** when written by cannabis-intoxicated programmers.
- Finding: **Cannabis use decreases program correctness** ($0.0005 < p < 0.05$, $0.28 < d < 0.44$, 10-14% fewer passed tests). In particular, cannabis impairs the ability to write and trace through programs.

**RQ2:** How does cannabis intoxication while programming impact programming speed?

- Hypothesis: Cannabis-intoxicated programmers **will take longer to write** programs.
- Finding: **Cannabis use impairs programming speed** ($p < 0.04$, $d = 0.3$, 10-14% slower). This decrease in speed is associated with typing slower, deleting more characters, and more time spent not typing.
High vs. Sober: How does Cannabis Impair Programming?

Programming While Sober

- Normal Keystrokes
- Delete Keystrokes
- Finished with Correct Solution

1-D Array Problem (Sober)
High vs. Sober: How does Cannabis Impair Programming?

Programming While Sober

- Normal Keystrokes
- Delete Keystrokes
- Finished with Correct Solution

Programming While High

1-D Array Problem (Sober)

1-D Array Problem (High)
Lens 3 - Summary: Psychoactive Substances and Programming

- By surveying 800 programmers, we found that psychoactive substance use is common in software.

- Despite anecdotes to the contrary, we only observed evidence of cannabis impairing productivity.

- This work demonstrates the usefulness of objective measures and careful controlled experimental design to come to evidence-based conclusions on anecdotal software productivity factors.
1. FSE, 2024  
Can LLMs Transform Natural Language Intent into Formal Methods Postconditions?  
Endres, M., Fakhoury, S., Chakraborty, S., Lahiri, S.

2. ICSE, 2024a  
Causal Relationships and Programming Outcomes: A Transcranial Magnetic Stimulation Experiment,  

3. ICSE, 2024b  
High Expectations: An Observational Study of Programming and Cannabis Intoxication,  
He, W., Parikh, M., Weimer, W., Endres, M.

4. FSE, 2023  
A Four-Year Study of Student Contributions to OSS with a Lightweight Intervention,  
Fang, Z., Endres, M., Zimmermann, T., Ford, D., Weimer, W., Leach., K., Huang, Y (Distinguished Paper)

5. ICSE, 2023  
From Organizations to Individuals: Psychoactive Substance Use By Professional Programmers,  
Newman, K., Endres, M., Weimer, W., Johnson, B.

6. OOPSLA, 2022  
Seq2Parse: Neurosymbolic Parse Error Repair,  
Sakkas, G., Endres, M., Guo, P., Weimer, W., Jhala, R.

7. ICSE, 2022  
Hashing It Out: A Survey of Programmers’ Cannabis Usage, Perception, and Motivation,  
Endres, M., Boehnke, K., Weimer, W.

8. ICSE-SEET, 2022  
Debugging with Stack Overflow: Web Search Behavior in Novice and Expert Programmers,  
Li, A., Endres, M., Weimer,

9. FSE, 2021  
To Read or To Rotate? Comparing the Effects of Technical Reading Training and Spatial Skills Training...  
Endres, M., Fansher, M., Shah, P., Weimer, W.

10. ICSE, 2021  
Relating Reading, Visualization, and Coding for New Programmers: A Neuroimaging Study  
Endres, M., Karas, Z., Hu, Z., Kovelman, I., Weimer, W

11. SIGCSE, 2021  
An Analysis of Iterative and Recursive Problem Performance,  
Endres, M., Weimer, W., Kamil, A.

12. PLDI, 2020  
Type Error Feedback via Analytic Program Repair  
Sakkas, G., Endres, M.,Cosman, B.,Weimer, W.,Jhala, R.

13. SIGCSE, 2020  
Pablo: Helping Novices Debug Python Code Through Data-Driven Fault Localization  
Cosman, B., Endres, M., Sakkas, G., Medvinsky, L., Yao-Yuan,Y.,Jhala, R.,Chaudhuri, K.,Weimer, W.

14. ASE, 2019  
InFix: Automatically Repairing Novice Program Inputs  
Endres, M., Cosman, B., Sakkas, G., Jhala, R., Weimer, W.
Student Advisees

Yiannos Demetriou
BS CS, 2023

Wenxin He
MS CS, 2024

Annie Li
BS CS, 2022

Kaia Newman
BS CS, 2023

Manasvi Parikh
BS CS, 2023

Interdisciplinary Collaborators

Madison Fansher
Cognitive Psychology

Xiaosu (Frank) Hu
Developmental Psychology

Priti Shah
Cognitive Psychology

Kevin Bohenke
Anesthesiology and Chronic Pain

Ioulia Kovelman
Developmental Psychology
Developing Efficient and Usable Programming Support

Supporting non-traditional novices in writing more correct code faster

Designing Effective Developer Training

Use cognitive insights to inform training and improve programming outcomes

Understanding External Productivity Factors

Exploring how substance use impacts software productivity

A Human-Focused Approach to Improving Programmer Productivity

Madeline Endres, PhD Candidate, University of Michigan
Bonus Slides
Data Analysis Pipeline

● **Step 1: Preprocessing**
  ○ Raw fNIRs Data (light intensity levels) → optical density data (how much is being absorbed?)
  ○ Optical density data → HbO/HbR signal (oxygenated vs deoxygenated blood)

● **Step 2: Individual Modeling**
  ○ We model the hemodynamic response for each subject individually using a GLM
  ○ Quality control checks are used to filter noisy data (e.g., signal to noise ratio, anticorrelation of HbO and HbR, visual activation spot checking)

● **Step 3: Group Modeling**
  ○ Used a linear mixed effects models, contrasting Task > Baseline activations
  ○ We applied a false-discovery rate (FDR) correction (q < 0.05) to account for multiple comparisons
Results: Baseline Activation

Reading > Rest

Mental Rotation > Rest
Results: Baseline Activation

- **Reading > Rest**
- **Coding > Rest**
- **Mental Rotation > Rest**
Supporting Publications

1. **ICSE, 2024**
   Causal Relationships and Programming Outcomes: A Transcranial Magnetic Stimulation Experiment,
   Ahmad, H., **Endres, M.**, Newman, K., Santiesteban, P., Shedden, E., Weimer, W.

2. **ICSE, 2024**
   High Expectations: An Observational Study of Programming and Cannabis Intoxication,
   He, W., Parikh, M., Weimer, W., **Endres, M.**

3. **FSE, 2023**
   A Four-Year Study of Student Contributions to OSS with a Lightweight Intervention,
   Fang, Z., **Endres, M.**, Zimmermann, T., Ford, D., Weimer, W., Leach, K., Huang, Y

4. **ICSE, 2023**
   From Organizations to Individuals: Psychoactive Substance Use By Professional Programmers,
   Newman, K., **Endres, M.**, Weimer, W., Johnson, B.

5. **OOPSLA, 2022**
   Seq2Parse: Neurosymbolic Parse Error Repair, Sakkas, G., **Endres, M.**, Guo, P., Weimer, W., Jhala, R.

6. **ICSE, 2022**
   Hashing It Out: A Survey of Programmers’ Cannabis Usage, Perception, and Motivation,
   **Endres, M.**, Boehnke, K., Weimer, W.

7. **ICSE-SEET, 2022**
   Debugging with Stack Overflow: Web Search Behavior in Novice and Expert Programmers,
   Li, A., **Endres, M.**, Weimer, W.

8. **FSE, 2021**
   To Read or To Rotate? Comparing the Effects of Technical Reading Training and Spatial Skills Training...
   **Endres, M.**, Fansher, M., Shah, P., Weimer, W.

9. **ICSE, 2021**
   Relating Reading, Visualization, and Coding for New Programmers: A Neuroimaging Study

10. **SIGCSE, 2021**
   An Analysis of Iterative and Recursive Problem Performance, **Endres, M.**, Weimer, W., Kamil, A.

11. **PLDI, 2020**
    Type Error Feedback via Analytic Program Repair
    Sakkas, G., **Endres, M.**, Cosman, B., Weimer, W., Jhala, R.

12. **SIGCSE, 2020**
    Pablo: Helping Novices Debug Python Code Through Data-Driven Fault Localization

13. **ASE, 2019**
    InFix: Automatically Repairing Novice Program Inputs
    **Endres, M.**, Cosman, B., Sakkas, G., Jhala, R., Weimer, W.
```
1  u = 42
2  x = float(input())
3  print(x * math.e / 2)
```
Anecdotal evidence abounds:

Many programmers use cannabis while programming.
Cannabis use can conflict with corporate anti-drug policies

This conflict can lead to hiring shortages!

We have a strict drug and alcohol policy. Employees are not permitted to use, possess, sell, transfer, manufacture, distribute, or be under the influence of illegal drugs on Cisco-owned or leased property, during working hours, while on company business, or while using company property.

Although certain jurisdictions may allow the prescription or other use of marijuana, this policy also applies to marijuana, which remains illegal under U.S. Federal law. Employees are not permitted to use, possess, sell, transfer, manufacture, distribute or be under the influence of these drugs while on Cisco owned or leased property, during working hours, while on company business, or while using company property. In addition, no employee may report for work, go on or remain on duty while under the influence of, or impaired by, alcohol, or these drugs or substances.

We find that 29% of software developers have taken a drug test for a programming-related job.

The FBI Says It Can't Find Hackers to Hire Because They All Smoke Pot

The FBI is struggling to find good hackers because of marijuana rules.

By MARY SCHUMACHER
THE FRESH TOAST | APR 23, 2018 AT 11:52 AM
How can we represent ill-parsed programs when \textit{training} our classifier?

Buggy Program

\begin{verbatim}
def foo(a):
    return a + 42

def bar(a):
    b = foo(a) + 17
    return b +
\end{verbatim}

Token Sequence

\begin{verbatim}
def name(name): \
    return name + number \
    
    def name(name): \
    indent name = name(name) + number \
    return name + \
    dedent end_marker
\end{verbatim}
How can we represent ill-parsed programs when **training** our classifier?

**Buggy Program**

```python
def foo(a):
    return a + 42

def bar(a):
    b = foo(a) + 17
    return b +
```

**Token Sequence**

```python
def name(name): \n    indent return name + number \n    dedent \n
def name(name): \n    indent name = name(name) + number \n    return name + \n    dedent end_marker
```

**Abstracted Token Sequence**

```
Stmt \n
def name Params: \n    indent Stmt \n    return Expr BinOp \n    dedent end_marker
```

Great! But we have a new problem: **Ambiguity**

*each abstracted token sequence can lead to multiple different ECE parse trees!*
My Approach To Programming Productivity: What's Next?

The next generation of neurosymbolic productivity support

The continued use of medical imaging to inform programming practice

Using controlled experimental design and objective measures to turn anecdote into evidence
Using **Technical Reading Ability** as a lens for facilitating the ability to understand and communicate complex technical ideas at varying levels of abstraction.
My Approach To Programming Productivity: What's Next?

Using Technical Reading Ability as a lens for facilitating the ability to understand and communicate complex technical ideas at varying levels of abstraction.

Increasing participation and retention in computing for diverse programmer groups through improving developer wellbeing.
How can we represent ill-parsed programs when training our classifier?

Buggy Program

```
def foo(a):
    return a + 42

def bar(a):
    b = foo(a) + 17
    return b +
```

Token Sequence

```
def name(name): \
    indent return name + number \n
dedent \n
def name(name): \
    indent name = name(name) + number \nreturn name + \n
dedent end_marker
```

Abstracted Token Sequence

```
Stmt \n
def name Params: \
    indent Stmt \nreturn Expr BinOp \
dedent end Marker
```

Great! But we have a new problem: **Ambiguity**

**Solution:** Learn a *Probabilistic Context Free Grammar* to Pick the Right One

```
S  → Stmts end_marker (p = 100.0%)
Stmts → Stmt \n      | Stmt \nStmts (p = 61.23%)
Stmt → ExprStmt (p = 62.64%) | RetStmt (p = 7.59%) | ...
RetStmt → return (p = 1.61%) | return Args (p = 98.39%)
```
**Seq2Parse**: Efficient Fixes for Novice Parse Errors

- **Program With Parse Error**
- **Relevant Error Rule Predictor** (Sequence Classifier)
- **Error-Correcting Erley Parser**
- **Fixed Program**

---

**Python Tutor Dataset**

- **Fixed Programs**
- **Parse Error Programs**
Seq2Parse: Efficient Fixes for Novice Parse Errors

Program With Parse Error

Partial Parser

Relevant Error Rule Predictor (Sequence Classifier)

Error-Correcting Erley Parser

Fixed Program

Probabilistic Context Free Grammar

Python Tutor Dataset

Fixed Programs

Parse Error Programs

125
Cannabis sativa is the world’s most commonly used illicit substance, used by more than 192 million people in 2018.

Cannabis is used for many reasons both medical (e.g., pain relief) and recreational (e.g., altered consciousness).

Cannabis’s legality is changing rapidly with many countries (e.g., UK, Colombia, Canada, Malawi) recently taking steps towards legalization.
Self-reported subjective programming performance when high (compared to when sober)

- Much better: 4.3%
- Better: 12.9%
- Same/Cannot tell: 20.0%
- Much worse: 24.3%
- Worse: 31.4%
- Extremely worse: 7.1%
<table>
<thead>
<tr>
<th>Desired Research Attribute</th>
<th>Why I'm Excited</th>
</tr>
</thead>
<tbody>
<tr>
<td>Provide <em>Theoretically-Grounded</em> and <em>Actionable Insights</em></td>
<td>Bridging the gap between novel theoretical ideas to supporting programmers in practice leads to higher impact</td>
</tr>
<tr>
<td>Include <em>Empirical or Objective Measures</em> of Programmers</td>
<td>Captures aspects of programming beyond self-reporting alone, including unconscious behaviors and habits</td>
</tr>
<tr>
<td><em>Minimize Scientific Bias</em> to Support Generalizability</td>
<td>Controlled experimental design can capture a signal, even for complex human behavior</td>
</tr>
<tr>
<td>Support <em>Diverse Developers</em></td>
<td>I prefer approaches that not only help programmers in general, but also help those who need the most support</td>
</tr>
</tbody>
</table>
Objective measures of cognition

Models of Programming Cognition

Identify Relevant Cognitive Skills

Spatial Visualization

Technical Reading