Three Lenses for Improving Programmer Productivity

From Anecdote to Evidence

Madeline Endres, PhD Proposal, 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, Brianna Pritchett

#### A Large-Scale Survey on the Usability of AI Programming Assistants: Successes and Challenges

Jenny T. Liang, Chenyang Yang, 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>Provide <em>Theoretically-Grounded</em> and Actionable Insights</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>
</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>
<tr>
<td><strong>Minimize Scientific Bias to Support Generalizability</strong></td>
<td>Controlled experimental design can capture a signal, even for complex human behavior</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><strong>Minimize Scientific Bias to Support Generalizability</strong></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

1/2 have never taken a university course

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**

Write code in [Python 3.11](https://www.python.org) [newest version, latest features]

```
1 def listSum(numbers):
2     if not numbers:
3         return 0
4     else:
5         (f, rest) = numbers
6         return f + listSum(rest)
7
8 myList = (1, (2, (3, None)))
9 total = listSum(myList)
```
Parse Errors

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

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

```
SyntaxError: missing parentheses in call to print
```

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'
```
Parse Errors

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

Proposed Approach: Neurosymbolic technique, **Seq2Parse**

*Preliminary results 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

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

*Preliminary results 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**

*Preliminary results 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'

SyntaxError: missing parentheses in call to print

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

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

*Preliminary results 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**.

### Most Common Python Errors Faced By Novices

- **SyntaxError**: 77.40%
- **TypeError**: 13.60%
- **AttributeError**: 3.00%
- **IndexError**: 2.60%
- **ValueError**: 2.30%
- **Other Errors**: 1.10%

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

Symbolic Approach?

Neural Approach?
Parsing Overview

Program \( P \)

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

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

Grammar \( G \)

\[
\begin{align*}
S & \rightarrow \text{Stmts end\_marker} \\
\text{Stmts} & \rightarrow \text{Stmt} \ \text{n} \ | \ \text{Stmt} \ \text{n} \ \text{Stmts} \\
\text{Stmt} & \rightarrow \text{FuncDef} \ | \ \text{ExprStmt} \\
& \quad | \ \text{RetStmt} \ | \ \text{PassStmt} \ | \ ...
\text{FuncDef} & \rightarrow \text{def} \ \text{name} \ \text{Params} : \ \text{Block}
\text{Block} & \rightarrow \ \text{n} \ \text{indent} \ \text{Stmts} \ \text{dedent}
\end{align*}
\]
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
Stmt → Stmts | Stmts
Stmt → FuncDef | ExprStmt
| RetStmt | PassStmt | ...
FuncDef → def name Params : Block
Block → \n indent Stmts dedent
```

Diagram of parse tree:

```
FuncDef
    |       |
    def    name Params : Block
    |       |
    ( name ) \n indent Stmts dedent
    |       |
    Stmt \n ExprStmt Stmts \n
```
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 dendent
New_S  →  S  |  S Insert
RetStmt →  |  E_return  |  E_return Args
E_return →  return  |  ε  |  Replace
          |  Insert return
E_number →  number  |  ε  |  Replace
          |  Insert number
...```

Fixing Parse Errors: Error Correcting Earley Parsers
Fixing Parse Errors: Error Correcting Earley Parsers

Program $P$

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

def bar(a):
```

Too many rules!

Grammar $G'$

```latex
S \rightarrow \text{Stmts end_marker}

\text{Stmts} \rightarrow \text{Stmt \ n} \mid \text{Stmt \ n \ Stmts}

\text{Stmt} \rightarrow \text{FuncDef} \mid \text{ExprStmt}

\text{ExprStmt} \rightarrow \text{RetStmt} \mid \text{PassStmt} \mid \ldots

\text{FuncDef} \rightarrow \text{def name Params : Block}

\text{Block} \rightarrow \text{\ n indent Stmts dendent}

\text{New}_S \rightarrow S \mid S \ \text{Insert}

\text{RetStmt} \rightarrow \text{| E_return | E_return Args}

\text{E_return} \rightarrow \text{return} \mid \epsilon \mid \text{Replace}

\text{E_return} \rightarrow \text{Insert return}

\text{E_number} \rightarrow \text{number} \mid \epsilon \mid \text{Replace}

\text{E_number} \rightarrow \text{Insert number}

\ldots
```
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 quite 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 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 minimal parse error fix, but are slow in practice because they consider too many production rules, the vast majority of which are not needed to fix any given novice error.
- In contrast, Neural approaches are fast and can leverage historical 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
**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

Relevant Error Rule Predictor

26 Error-Correcting Earley Parser
**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

---

27
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:** We trained multiple model variations, including one that **outputs the 20 most likely error production rules** for a given Buggy Program
  - These rules are then fed into the Error Correcting Earley Parser
Preliminary results: 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.
Preliminary results: 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
Preliminary results: 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!
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
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.

_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.
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 so far?

- 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 so far?

- 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><strong>Floyd et al., (2017)</strong></td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>Huang et al., (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.

### What do we know so far?

- Neuroimaging
  - Experiment
  - Is programming like Reading?
  - Is programming like Spatial Reasoning?
  - What about with novices?

<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 <em>et al.</em>, (2014)</td>
<td>✔</td>
<td></td>
<td>?</td>
</tr>
<tr>
<td>Siegmund <em>et al.</em>, (2017)</td>
<td>✔</td>
<td></td>
<td>?</td>
</tr>
<tr>
<td>Floyd <em>et al.</em>, (2017)</td>
<td>✔</td>
<td></td>
<td>?</td>
</tr>
<tr>
<td>Huang <em>et al.</em>, (2019)</td>
<td></td>
<td>✔</td>
<td>?</td>
</tr>
</tbody>
</table>
Proposed Study Overview

Phase 1: **Neuroimaging**

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

Phase 2: **Transfer Training**

- We propose to investigate the usefulness of transfer training in computing comparing the impact of two cognitive interventions on novice programming performance in a controlled, longitudinal study.
Proposed Study Overview

Phase 1: **Neuroimaging**

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

Phase 2: **Transfer Training**

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

*ICSE, 2021*

*FSE, 2021*
Phase 1: Neuroimaging Method

- We propose using **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 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 semester

Week 4-5.5: Brain scans

Week 3: Participant recruitment from CS1

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

Please type the corresponding letter which best represents the return value of the function call below:

```cpp
bool func(bool x, bool y) {
  return (x && y) || (x && !y);
}

func(true, false)
```

Please type the corresponding letter of the bottom objects to give your answer:

- A
- B
We compare brain activation during three tasks:

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

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

- 36 participants, **31 valid** (24 female, 7 male)

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

- **Comparative Activation**: Do true programming novices rely more on spatial or language brain regions while programming?
  - a. How do novices' brain activation patterns compare to those of expert developers?
Preliminary Results: Comparative Brain Activation

- **Question:** Do novices use more spatial 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.
Preliminary Results: Comparative Brain Activation

- **Question:** Do novices rely more on spatial 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.

![Brain Activation Images]

- Coding > Reading
- Coding > Mental Rotation
Preliminary Results: Comparative Brain Activation

- **Question:** Do novices rely more on spatial 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.
Preliminary Results: Comparative Brain Activation

- **Question:** Do novices rely more on spatial 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.
Preliminary Results: Comparative Brain Activation

● Question: **Do novices rely more on spatial 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**.
Preliminary Results: Comparative Brain Activation

- **Question**: Do novices rely more on spatial 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, programming looks more like spatial visualization than like reading. Now what?
Preliminary 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 continues to novices. For less experienced programmers, programming and reading show less 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 "more similar" 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 propose an intervention to teach **strategies** for **efficiently understanding scientific writing**

- Strategies focused on **using structural cues to scan texts** to retrieve and understand key points
  - Experienced programmers tend to read code non-linearly, focusing on high level features.
Semester CS1 Course With Final Exam
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?

![Graph showing code completion questions over time with pre-test and post-test scores for reading and spatial skills](image-url)
How did the Reading Training Help?

- Code Completion Questions
  - Score vs. Time (Pre-test vs. Post-test)
  - Reading (red), Spatial (blue)

- Definitional Questions
  - Score vs. Time (Pre-test vs. Post-test)
  - Reading (red), Spatial (blue)
How did the Reading Training Help?

![Graphs showing the improvement in scores over time for Code Completion Questions, Definitional Questions, and Tracing Questions after reading and spatial training.](image-url)
How did the Reading Training Help?

- Code Completion Questions
- Definitional Questions
- Tracing Questions

$p = 0.03$
Lens 2 Summary

Phase 1: **Neuroimaging**

- *Proposal*: model novice programmer cognition using using fNIRs
- *Preliminary Results*: For novices, programming is a challenging *working-memory intensive* task. In contrast to studies with experts, *spatial reasoning is "more similar" to programming than reading* for novices at a cognitive level

Phase 2: **Transfer Training**

- *Proposal*: investigate transfer training in computing by *comparing the impact of two cognitive interventions on novice programming* in a controlled, longitudinal study
- *Preliminary Results*: *Technical Reading Training helped programming ability more than spatial training*, 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, 2023, 2024

Credit: XKCD Comic, https://xkcd.com/323/
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.

"Taking LSD was a profound experience, one of the most important things in my life”
- Steve Jobs

Coder’s High
Programming is just like drugs, except the dealer pays you.

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

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, they cannot bring marijuana on company premises or company-related travel.

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.
Proposed Study Overview

Phase 1: **Interviews and Survey**

- We propose to understand if, when, or why developers use psychoactive substances while programming using a *large-scale survey* and *qualitative interviews* with professional prop

Phase 2: **Observational Study**

- We propose to build a mathematical model of how one substance, cannabis, impacts programming performance using a *controlled, observational study*.

*ICSE, 2022, 2023*

*Preliminary work not published*
Phase 1: **Survey Methodology**

- **Goal:** To understand *if, when, and why* developers use cannabis while programming

- **Survey Design:**
  - 15-minute Qualtrics survey with questions about demographics, programming background, cannabis usage history, and experiences using cannabis while programming

- **Survey Populations:**
  - **GitHub:** Sent emails to 5,000+ US-based developers on popular projects
  - **University of Michigan:** Sent emails to 5,000+ current and recent graduates

- **Survey Ethics:**
  - We also need to make sure we distribute this survey *ethically*: responses are *anonymous* and *confidential*
Phase 1: SURVEY RESPONSES

803 valid responses:
○ 440 from GitHub
○ 339 from University of Michigan
○ 24 from Social Media

Demographics:
○ 83% Men, 14% Women, 2% Non-binary
○ Ages range from 15 to 70
○ 56% Full-time programmers, 36% Students

Job Title (could select multiple)

<table>
<thead>
<tr>
<th>Job Title</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software Engineer</td>
<td>311</td>
</tr>
<tr>
<td>Developer</td>
<td>270</td>
</tr>
<tr>
<td>Systems Engineer</td>
<td>72</td>
</tr>
<tr>
<td>CS Researcher</td>
<td>53</td>
</tr>
<tr>
<td>CS Instructor</td>
<td>49</td>
</tr>
<tr>
<td>Data Scientist</td>
<td>49</td>
</tr>
</tbody>
</table>
Phase 1 Preliminary Results: Summary

Usage While Programming in Last Year

<table>
<thead>
<tr>
<th>Substance</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alcohol</td>
<td>24.53%</td>
</tr>
<tr>
<td>Cannabis</td>
<td>24.40%</td>
</tr>
<tr>
<td>Tobacco</td>
<td>5.73%</td>
</tr>
<tr>
<td>Amphetamines</td>
<td>4.73%</td>
</tr>
<tr>
<td>Hallucinogens</td>
<td>2.12%</td>
</tr>
</tbody>
</table>
Phase 1 Preliminary Results: **Summary**

Usage While Programming in Last Year

- **33%** use cannabis for work-related tasks
- **11%** use cannabis at a frequency likely to be caught by a drug test
- Qualitative evidence from cannabis-using includes conflicting experiences, with some reporting impairment with others reporting programming enhancement.
Phase 2: **A Controlled Study of Cannabis's Impacts**

- **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

```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(s: str) -> str:
        return """" # Participant implementation goes here"

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

PROBLEMS | OUTPUT | DEBUG CONSOLE | TERMINAL | PORTS | COMMENTS
---------|--------|---------------|----------|-------|-----------

Codespaces $ main* | 0△0 | 9 | Ln 21, Col 5 | Spaces: 4 | UTF-8 | CRLF | Python
Remote Programming Session 1

Programming Sober

Programming Intoxicated

Remote Programming Session 2
Remote Programming Session 1

Remote Programming Session 2


```c
class Solution:

    def removeDuplicates(self, s: str) -> str:
        stack = []

        for c in s:
            if stack and c == stack[-1]:
                stack.pop()
            else:
                stack.append(c)

        return ''.join(stack)
```

Remote Programming Distributed
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.
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.

Current Status:
We have received IRB approval for our proposed study and have funding for participants. We have successfully obtained preliminary data from 74 participants.
Lens 3 - Summary: Psychoactive Substances and Programming

- In a survey of 800 programmers, we found that psychoactive substance use is common in software, especially alcohol and cannabis.

- We found that many programmers use cannabis at rates that can be tested by current software drug policies, and that there are conflicting qualitative experiences of its impacts.

- We have received IRB approval to conduct an observational study of cannabis's impact on programmers, and have collected preliminary data.
Professional Programmers

ICSE 2023

ICSE 2022

ICSE 2024b

FSE 2023

ICSE-SEET 2022

ICSE 2024a

More Theoretical

PLDI 2020

FSE 2023

SIGCSE 2021

OOPSLA 2022

More Empirical

OOPSLA 2022

FSE 2021

SIGCSE 2020

ASE 2019

Programming Novices

ICSE 2021

ICSE 2020

SIGCSE 2020

103
<table>
<thead>
<tr>
<th>No.</th>
<th>Conference</th>
<th>Title</th>
<th>Authors</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>FSE, 2023</td>
<td>A Four-Year Study of Student Contributions to OSS with a Lightweight Intervention</td>
<td>Fang, Z., Endres, M., Zimmermann, T., Ford, D., Weimer, W., Leach, K., Huang, Y</td>
</tr>
<tr>
<td>3</td>
<td>ICSE, 2023</td>
<td>From Organizations to Individuals: Psychoactive Substance Use By Professional Programmers</td>
<td>Newman, K., Endres, M., Weimer, W., Johnson, B.</td>
</tr>
<tr>
<td>4</td>
<td>OOPSLA, 2022</td>
<td>Seq2Parse: Neurosymbolic Parse Error Repair</td>
<td>Sakkas, G., Endres, M., Guo, P., Weimer, W., Jhala, R.</td>
</tr>
<tr>
<td>5</td>
<td>ICSE, 2022</td>
<td>Hashing It Out: A Survey of Programmers’ Cannabis Usage, Perception, and Motivation</td>
<td>Endres, M., Boehnke, K., Weimer, W.</td>
</tr>
<tr>
<td>7</td>
<td>FSE, 2021</td>
<td>To Read or To Rotate? Comparing the Effects of Technical Reading Training and Spatial Skills Training...</td>
<td>Endres, M., Fansher, M., Shah, P., Weimer, W.</td>
</tr>
<tr>
<td>9</td>
<td>SIGCSE, 2021</td>
<td>An Analysis of Iterative and Recursive Problem Performance</td>
<td>Endres, M., Weimer, W., Kamil, A.</td>
</tr>
<tr>
<td>10</td>
<td>PLDI, 2020</td>
<td>Type Error Feedback via Analytic Program Repair</td>
<td>Sakkas, G., Endres, M., Cosman, B., Weimer, W., Jhala, R.</td>
</tr>
<tr>
<td>12</td>
<td>ASE, 2019</td>
<td>InFix: Automatically Repairing Novice Program Inputs</td>
<td>Endres, M., Cosman, B., Sakkas, G., Jhala, R., Weimer, W.</td>
</tr>
</tbody>
</table>
A Human-Focused Approach to Improving Programmer Productivity

Madeline Endres, PhD Candidate, University of Michigan
Bonus Slides
```python
u = 42
x = float(input())
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 \texttt{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): 
    return name + number 

def name(name):
    indent return name + number 
    return name + 
    dedent end_marker
```
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  
dedent

def name(name):  
    indent name = name(name) + number  
    return name + 
    dedent
```

Abstracted Token Sequence

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

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

each abstracted token sequence can lead to multiple different ECE parse trees!
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):  
    indent return name + number  
    dedent  

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

Abstracted Token Sequence

```python
Stmt  

def name Params:  
    indent Stmt  
    return 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 (p = 38.77%) | Stmt \n Stmts (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
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 x Parse Error Programs
**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**.
RQ2: Prediction

- **Question:** Can brain activation patterns at the start of CS1 predict future programming ability?

- **Method:** Correlate brain activity interactions with scores on a programming test at the end of the semester (11-12 weeks after the initial brain scan).

- **Result:** Yes, it is possible!

- Less-similar patterns of activation for coding and mental rotation in the right frontal hemisphere at the start of the semester predict better outcomes on the end-of-semester programming assessment ($r = -0.482$, $p = 0.006$).
RQ2: Prediction Implications

- Perhaps novices who transition away from general spatial skills to reading-associated domain-specific strategies earlier make more progress.

- Provides impetus for earlier pedagogical interventions.

- Note: we do not see our result supporting essentialist-based theories of programming ability.
  - Rather, it provides insight for more effectively understanding and removing computing barriers.
RQ2: Prediction Summary

Novice brain activity when programming can predict future programming ability.

Provides another window into understanding and ameliorating computing barriers.