Accessible Software Development

05-499/899 Fall 2024

Celebrating Accessibility

https://cmu-05-499.github.io

Andrew Begel and Patrick Carrington

Administrivia

- The election is over. The world will go on.
- P4 – Project Milestone 1 due next week, Thursday November 14 11:59pm.
- Talk to your TAs and let them know how things are going.
- If you are blocked on progress for any reason, reach out to your TAs and they will brainstorm with you how to work around it.

Accessible Software Engineering

Two meanings:

1. Supporting software development by people with disabilities
2. Developing software to be used by people with disabilities

Accessible Software Engineering

Two meanings:

1. Supporting software development by people with disabilities
2. Developing software to be used by people with disabilities

Deep Dive

- Programming by Voice
- Mixed Ability Collaboration in Programming

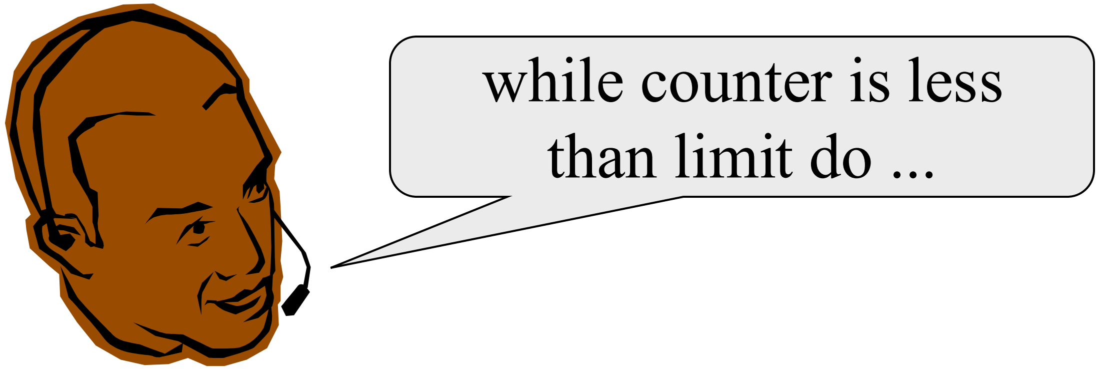
- Both by Andrew Begel (and his collaborators)!

Programming by Voice - Motivation

```
while (counter < limit) {
    ⍓
}
```

- Programmers conventionally use keyboard
  - Long hours at keyboard leads to higher risk of RSI
- Can speech-based programming be an alternative?
- Combines an unambiguous domain (programming) with an inherently ambiguous input modality (speech)
  - Great for exploring ambiguity handling in a new context

Programming by Voice



- My Goal
  - Find out how developers use code verbally. Use this to develop a naturally verbalizable input form.
  - Build development environment that supports verbal authoring, navigation, modification.
    - Extend conventional compiler analyses to support ambiguities generated by speech.
  - Learn how developers can use voice-based programming, and iterate design.

Challenges

□ Programming languages were not designed to be spoken.

□ Speech is inherently ambiguous. Programming tools were not designed for ambiguity.

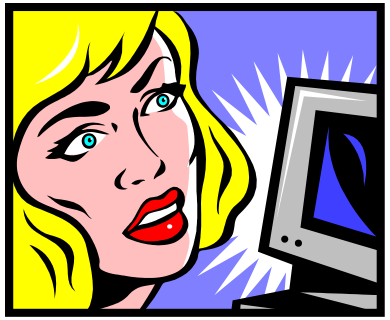
□ Speech tools are poorly suited for programming tasks.

□ Programmers are not used to verbal software development.

Talk Outline

- Introduction and Motivation
- ➢ **Programming by Voice**
- Program Analyses for Ambiguous Inputs
- Program Navigation and Editing
- Conclusion

Programming by Voice

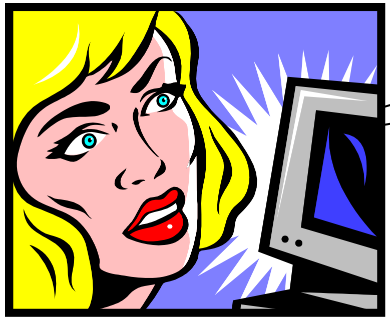


```
for (int i = 0; i < 10; i++ ) {
    ▌
}
```

Current Tools are Awkward!

VoiceCode
[Desilets 2004]

for loop … after left paren … declare india of type integer … assign zero … after semi … recall one … less than ten … after semi … recall one … increment … after left brace



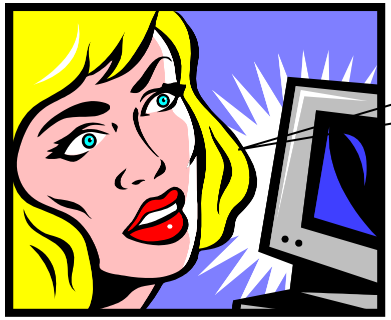
1
2
3
4

```
for (▌ ; ; ) {


}
```

Current Tools are Awkward!

VoiceCode
[Desilets 2004]

for loop … after left paren … declare india of type integer … assign zero … after semi … recall one … less than ten … after semi … recall one … increment … after left brace



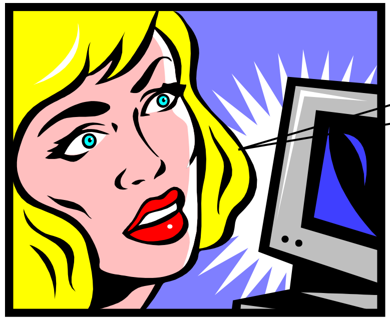
1	i
2	
3	
4	

```
for (int i = 0;  ; ) {

}
```

Current Tools are Awkward!

VoiceCode
[Desilets 2004]

for loop … after left paren … declare india of type integer … assign zero … after semi … recall one … less than ten … after semi … recall one … increment … after left brace



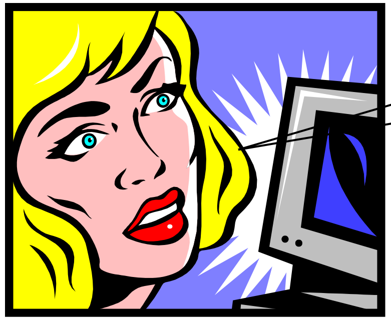
1	i
2	
3	
4	

```
for (int i = 0; i < 10;▌ ) {


}
```

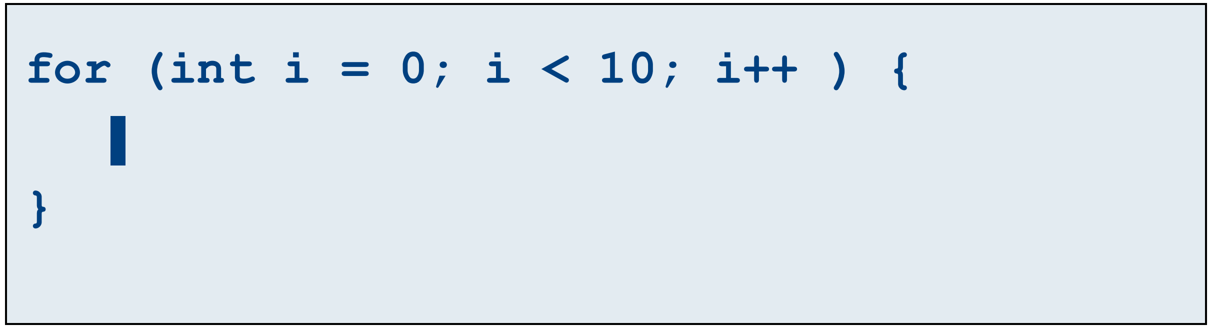
Current Tools are Awkward!

VoiceCode
[Desilets 2004]

for loop … after left paren … declare india of type integer … assign zero … after semi … recall one … less than ten … after semi … recall one … increment … after left brace

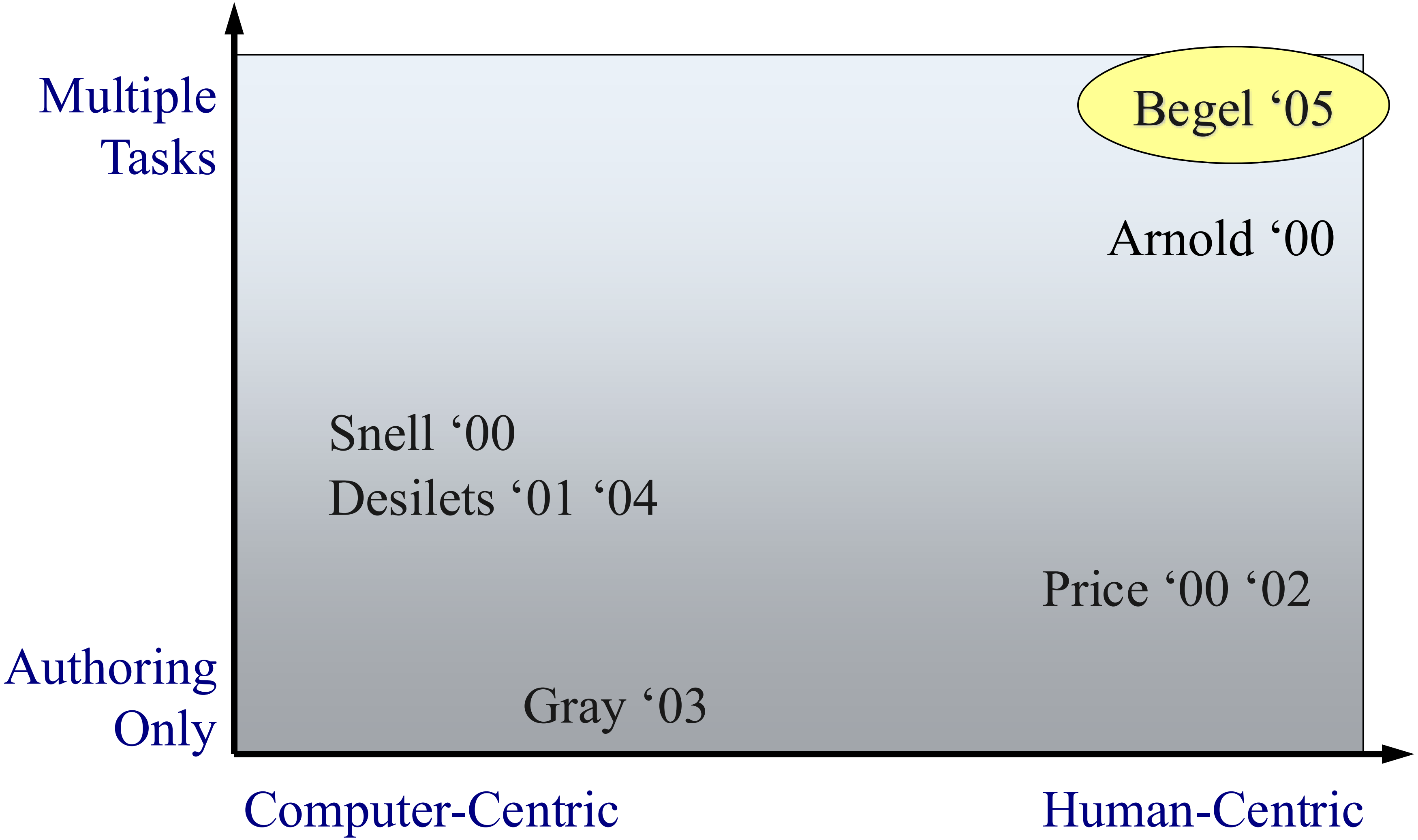


1	i
2	
3	
4	



```
for (int i = 0; i < 10; i++ ) {
   ▌
}
```

Programming by Voice Related Work



How do Programmers Speak Code?

- 10 programmers read Java code out loud
- Most programmers spoke the same way
- But, there were some interesting differences...

How do Programmers Speak Code?

Awkwardness by Design (Structural)

```
(int)foo
```

```
(3 + 5) * 7
```

How do Programmers Speak Code?

Individual Inconsistency

`System.out.println` vs.
`System out println`

`bar sub i` vs.
`bar of i` vs.
`i from bar`

How do Programmers Speak Code?

Native English speakers vs.
non-native speakers (Pronunciation)

`tur` vs. `t u r`

`println`

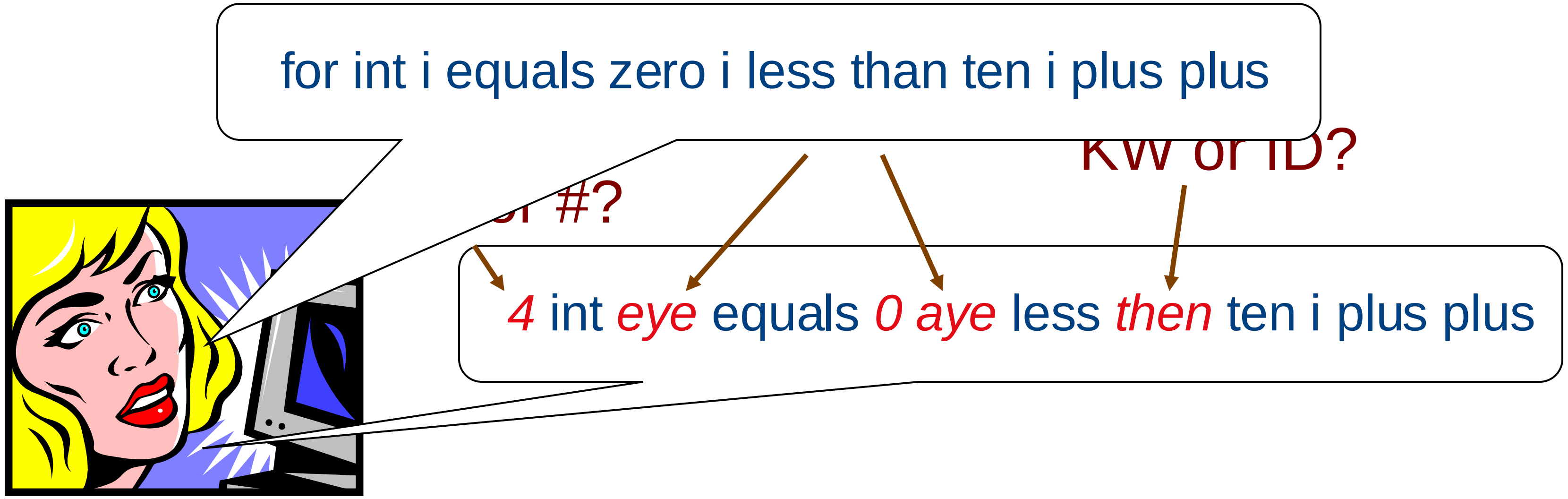
`array[i++]` vs. `array[i]++`

A More Natural Way to Code

for int i equals zero i less than ten i plus plus

```
for (int i = 0; i < 10; i++ ) {

}
```

Too Many Ambiguities



for int i equals zero i less than ten i plus plus

or #?

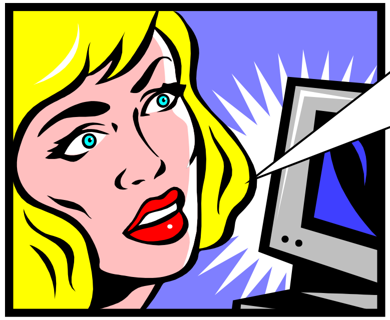
KW or ID?

4 int *eye* equals *0* *aye* less *then* ten i plus plus

```
for (int i = 0; i < 10; i++ ) {
    ▌
}
```

Sometimes It’s Non-Obvious

for times equals 8 file 2 load times equals one

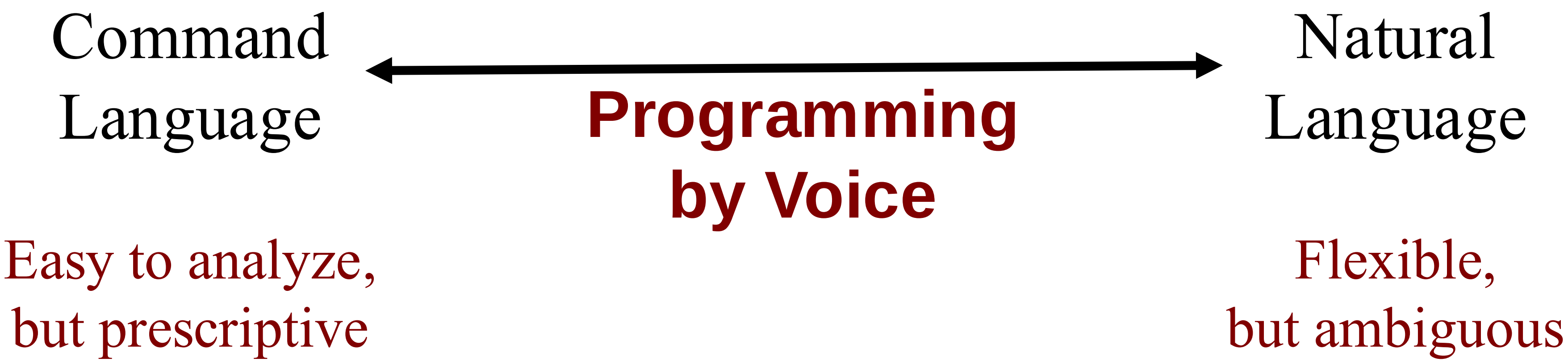


```
for (times = 8; file(2, load); times == one) {
  
}
```

```
fore *= 8; file.tooLode.times = won
```

```
4; times = ate(file).to(load).equals(1)
```

Design Tradeoffs



Command Language

Easy to analyze, but prescriptive

Programming by Voice

Natural Language

Flexible, but ambiguous

Spoken Java

- Semantically identical to Java
- Syntactically easier to say than Java
  - – Methodology generalizable to any computer language

1. All punctuation has English equivalents
   - Open Brace, End For Loop
2. Most punctuation is optional
3. Provide verbalization for all abbreviations
4. Relaxed phrasing for better fit with English
   - `(int)foo` □ "cast foo to integer"
   - `foo = 6` □ "set foo to 6"
   - `foo[i]++` □ "increment the ith element of array foo"

SPEED: Speech Editor

- Build an editor that supports *naturally* verbalized programs
  - SPEED: SPEech EDitor
    - Based on IBM ViaVoice, Eclipse IDE
  - Spoken Java Language for Composition
  - Spoken Command language for Navigation, Editing, Template instantiation, Refactorings, Search
  - Audible and visual feedback
    - Similar to JavaSpeak [Smith 2000]

Study – SPEech EDitor Usability

Goal: Understand how SPEED can be used by expert programmers

Hypothesis: SPEED is learnable and usable for standard programming tasks

1. Train 5 expert Java programmers on SPEED (20 minutes)
2. Create and modify code (30 minutes)
   - Build a Linked List data structure with associated algorithms

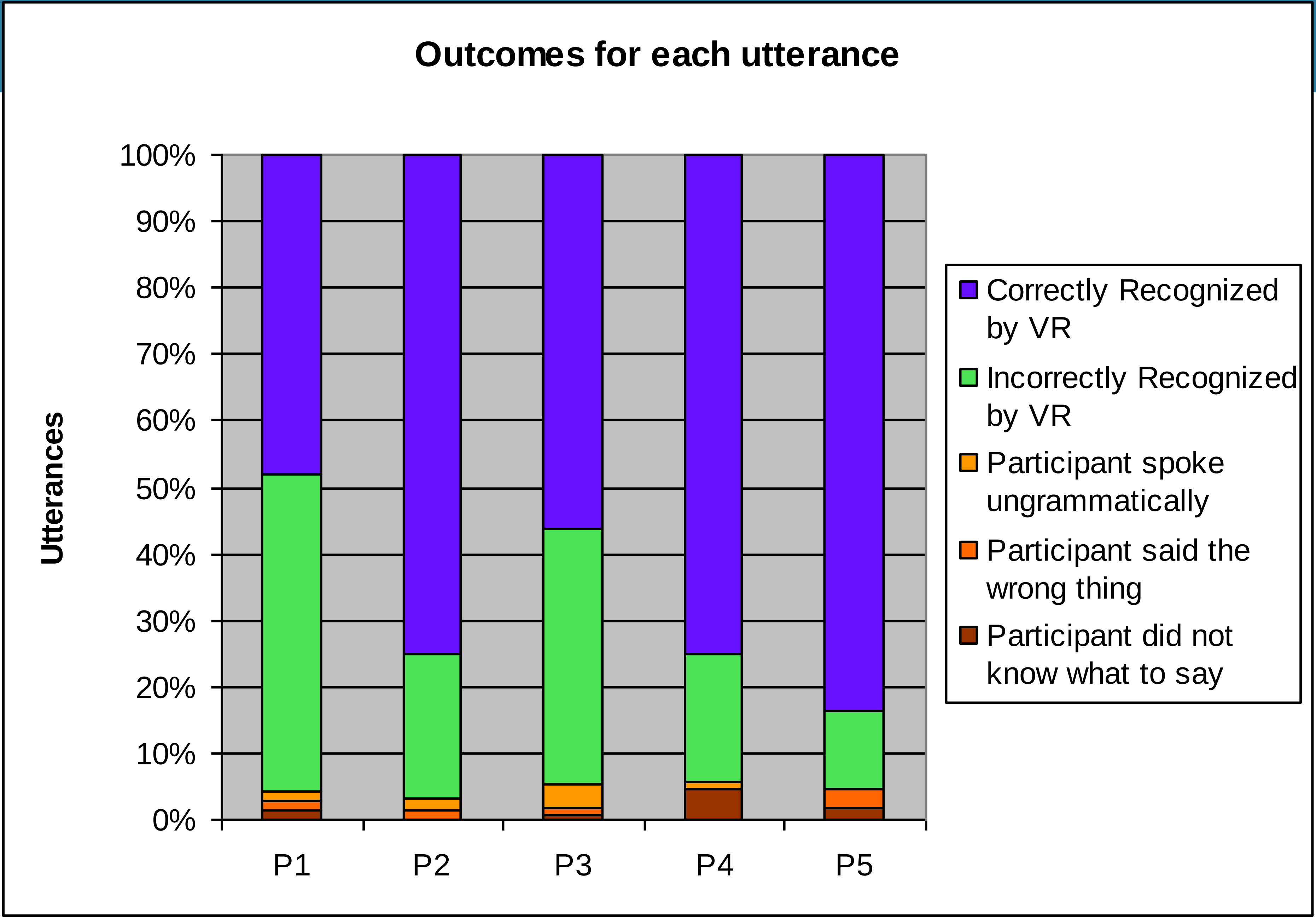
- 3 programmers used commercial speech recognizer
  2 programmers used *human* speech recognizer

Watch it in Action!

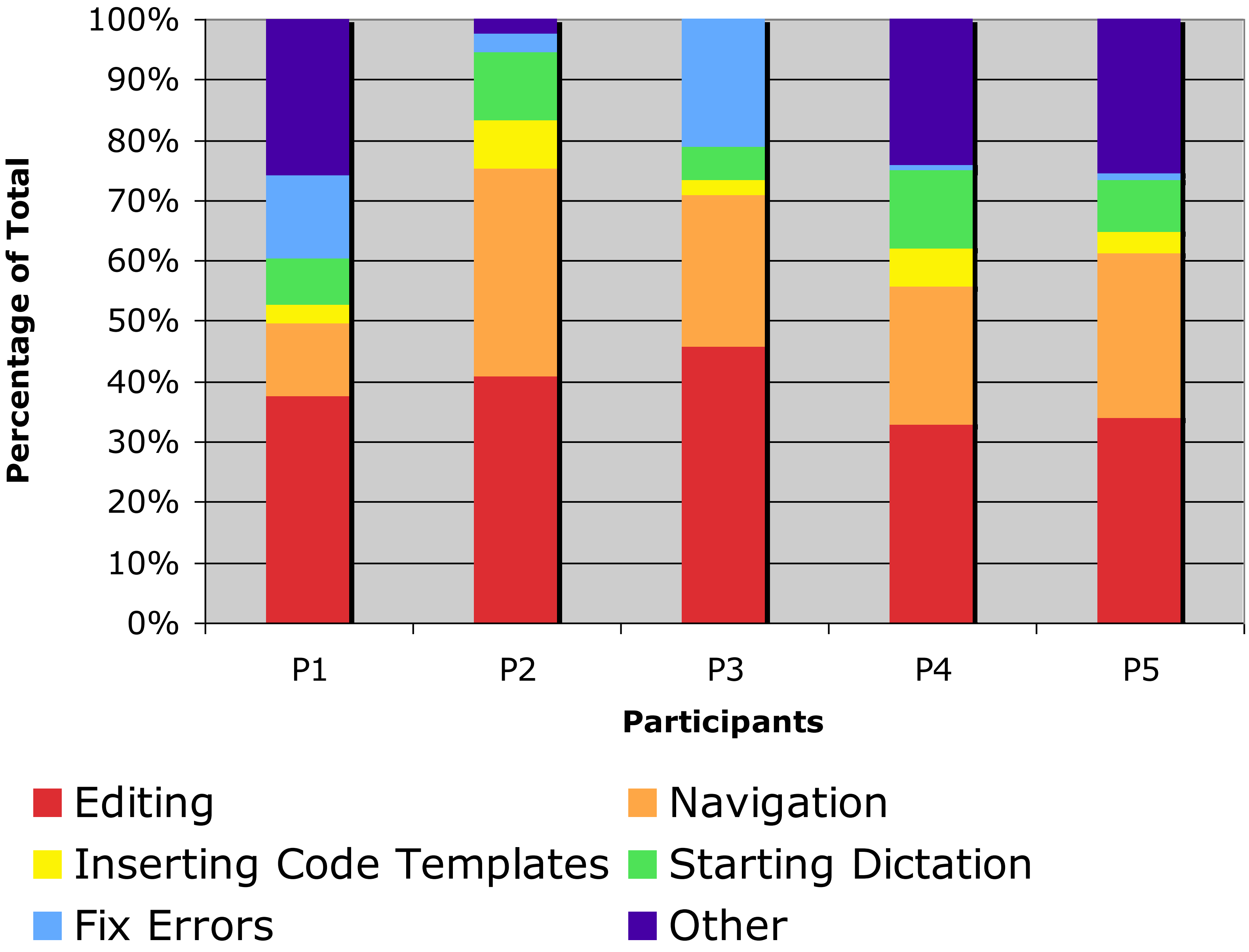
- https://andrewbegel.com/papers/dissertation-highlights.wmv

Metrics

- Number of Commands/Dictations Uttered vs. Recognized
- Number of Correctly Interpreted Recognition Events
- Features Used
  - Code Templates, Dictation, Navigation, Editing, Fixing Mistakes
- Quantity and Kinds of Mistakes
  - Speech Recognition, SPEED, User


Outcomes for each utterance
Utterances
100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%
P1
P2
P3
P4
P5
Correctly Recognized by VR
Incorrectly Recognized by VR
Participant spoke ungrammatically
Participant said the wrong thing
Participant did not know what to say

Correct Commands and Dictation



Summary of Results

- Commands were easy to learn and remember.
  - Very few user mistakes
- Most commands spoken for editing.
  - GOMS analysis predicts speech will be slower unless you can get a lot of text for each utterance.
  - Code templates provide “most bang for your buck”.
- Speakers were apprehensive about speaking code instead of describing it via code templates.

Paper Discussion

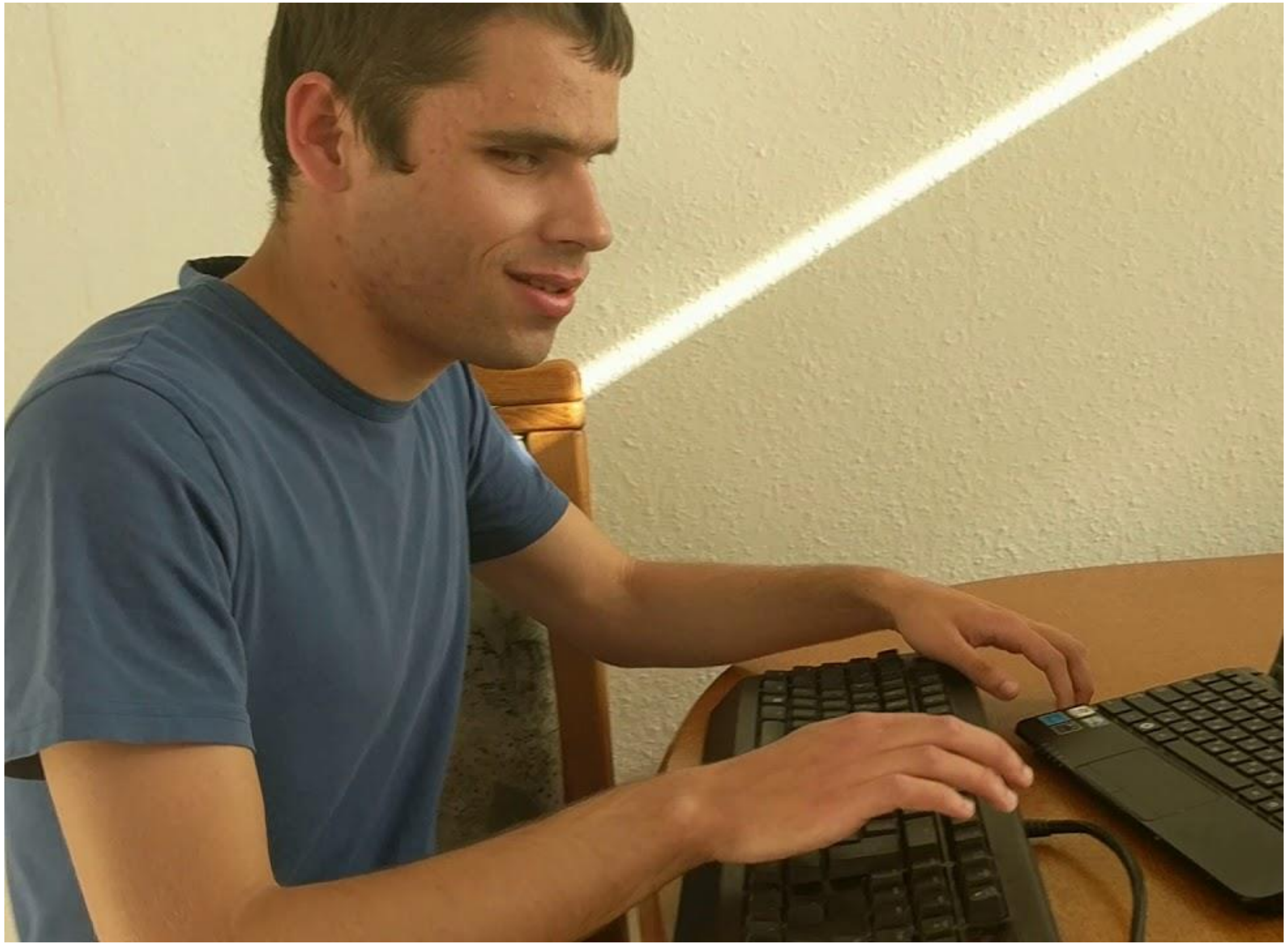
- Blocks4All Programming Environment
- Interviews and Observation of Blind Software Developers at Work to Understand Code Navigation Challenges
- Vocal Programming for People with Upper-Body Motor Impairments
- Spoken Programs

Meet Michael

Michael is blind.

He uses a screen reader to interact with his laptop.

He is a software developer who works on a team with 5 sighted developers.



In person collaboration

- Before COVID-19, Michael collaborated with others in person at the office.
- To walk through code together, Michael used his screen reader while his colleagues looked over his shoulder at the screen.

Synchronous, remote collaboration tools are inaccessible

- After COVID, everyone worked from home and used Zoom to share their screens and talk about the code out loud.
- Michael wanted to continue reading the code together with his team, but none of the remote collaboration tools were accessible.
  - Shared screens are completely inaccessible.
  - Co-editing IDEs only show everyone’s cursors and edit actions *visually*.
- Michael had to constantly remind his colleagues to say code locations out loud in order to stay in sync.
  - Sometimes, he just gave up and handed control of his computer to a remote colleague.

How do accessibility limitations impact developers with visual impairments in the workplace?

Agenda

1. Study of collaboration barriers experiences by 12 blind or visually impaired software developers
2. CodeWalk: an IDE with mixed ability collaboration support through sound effects and speech

In collaboration with Venkatesh Potluri (U. Washington), Maulishree Pandey (U. Michigan), Michael Barnett (Microsoft), and Scott Reitherman (Microsoft)

Study Details

- Semi-structured interviews with 12 blind and visual impaired software developers from USA and India
  - Aged 20-57 (avg. 32, median 26)
  - 10 male, 2 female
  - 6 blind, 3 visually impaired, 2 low vision, 1 legally blind
  - 8 employed in software, 4 students
  - 10 used screen readers, 2 used screen magnifiers
  - All worked with sighted colleagues

Collaborative Coding Activities

- Writing (co-editing)
- Debugging
- Code walkthroughs
- Code reviews
- Giving someone advice about the code
- Searching for examples
- Hackathons

Coding Tools

Screen Readers

- JAWS
- NVDA
- Voiceover
- Narrator
- Talkback

IDEs

- Visual Studio
- VS Code
- Notepad
- Notepad++
- CLI
- Eclipse

Everyone used multiple IDEs and screen readers

Tool Barriers

- Installation and setup processes could be inaccessible, even if the tool was itself accessible.
- IDEs and tools are *incompletely* compatible with screen readers.
- Software redesigns often break useful accessibility features.
- The team's engineering practices restrict choice for disabled developers.

Pandey, Maulishree, et al. "Understanding Accessibility and Collaboration in Programming for People with Visual Impairments." Proceedings of the ACM on Human-Computer Interaction 5.CSCW1 (2021): 1-30.

Asymmetric Collaboration Barriers

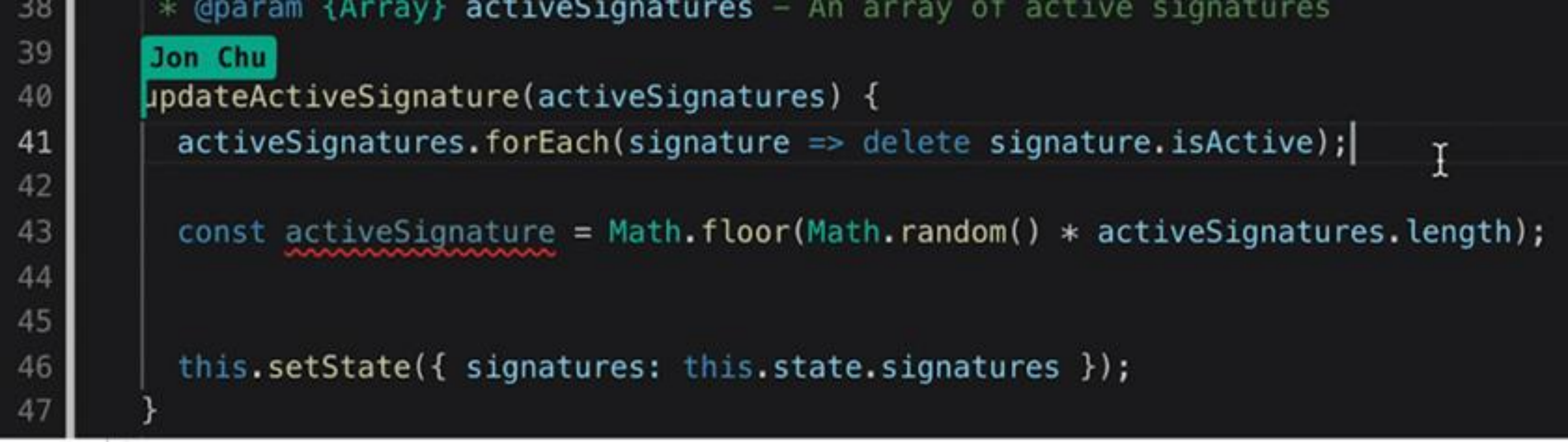
- Code sharing via screen sharing is inaccessible.
- Sighted colleagues forget to explain where they are in the code base, preventing blind developers from following along.
- It is hard to follow spoken details about code that blind developers have not yet read through themselves.
  - When blind developers get lost, they are hesitant to ask questions.
- Sighted colleagues get frustrated and take over the blind developer's computer.

CodeWalk: Shared Awareness in Mixed Ability Collaborative Software Development

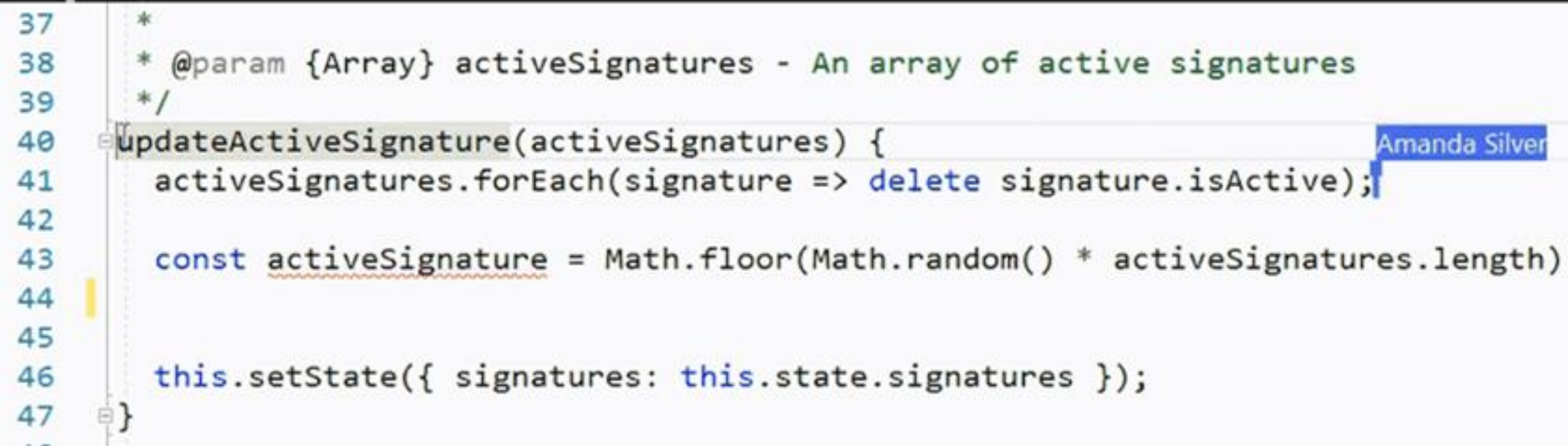
In collaboration with Venaktesh Potluri (U Washington), Maulishree Pandey (U Michigan), Michael Barnett (Microsoft), Scott Reitherman (Microsoft)

Co-editing support in IDEs

- Several IDEs now allow programmers to coedit and co-debug code from their respective IDEs
- They also offer "Follow" or "Observer" modes that allow collaborators to sync viewports



```
 * @param {Array} activeSignatures - An array of active signatures

updateActiveSignature(activeSignatures) {
  activeSignatures.forEach(signature => delete signature.isActive);

  const activeSignature = Math.floor(Math.random() * activeSignatures.length);


  this.setState({ signatures: this.state.signatures });
}
```



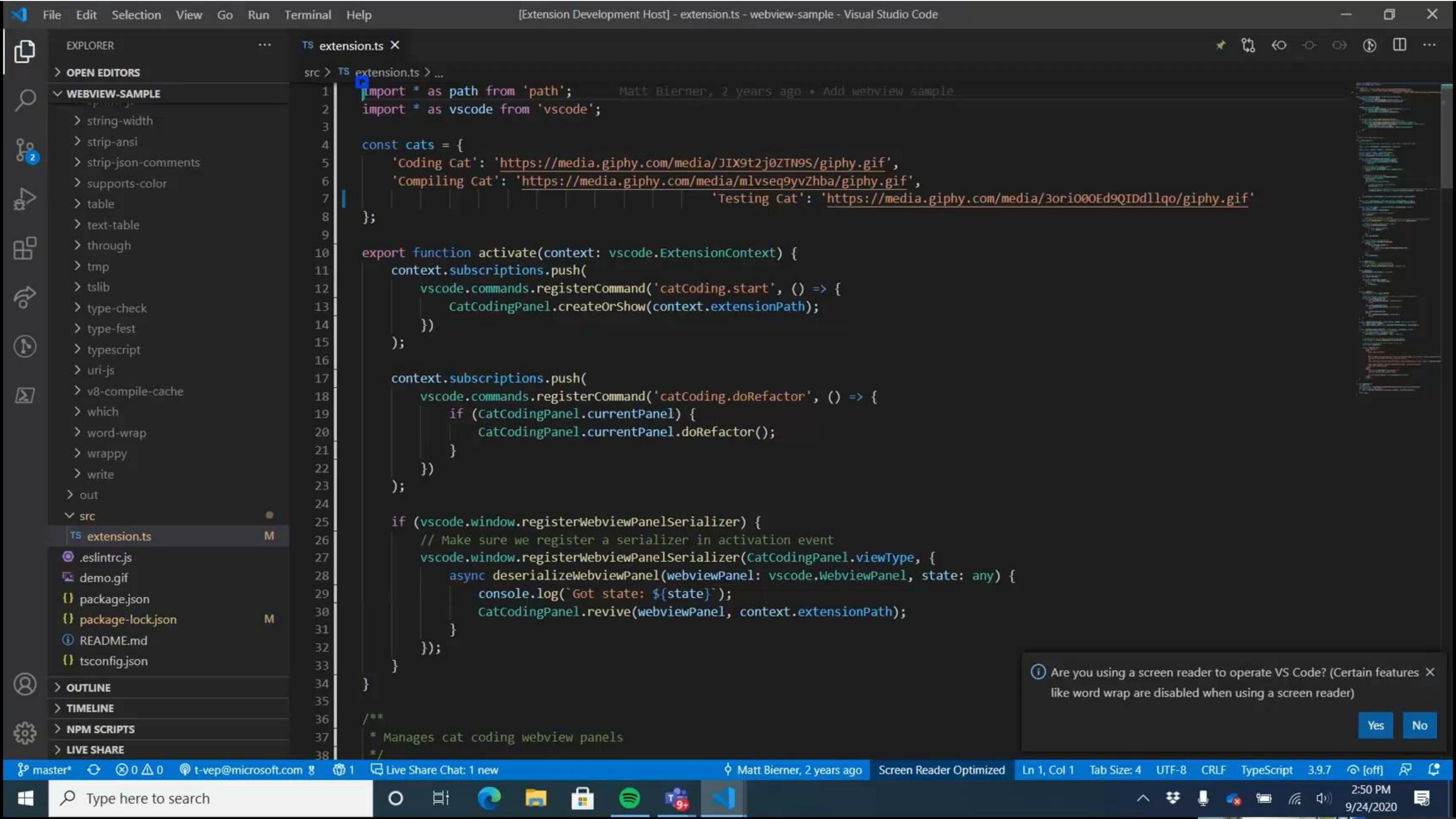
```
 *
 * @param {Array} activeSignatures - An array of active signatures
 */
updateActiveSignature(activeSignatures) {
  activeSignatures.forEach(signature => delete signature.isActive);

  const activeSignature = Math.floor(Math.random() * activeSignatures.length)


  this.setState({ signatures: this.state.signatures });
}
```

Co-editing support in IDEs

- Several IDEs now allow programmers to coedit and co-debug code from their respective IDEs
- They also offer "Follow" or "Observer" modes that allow collaborators to sync viewports
- The visual nature of these awareness cues makes them inaccessible to blind and visually impaired developers

Design Goals

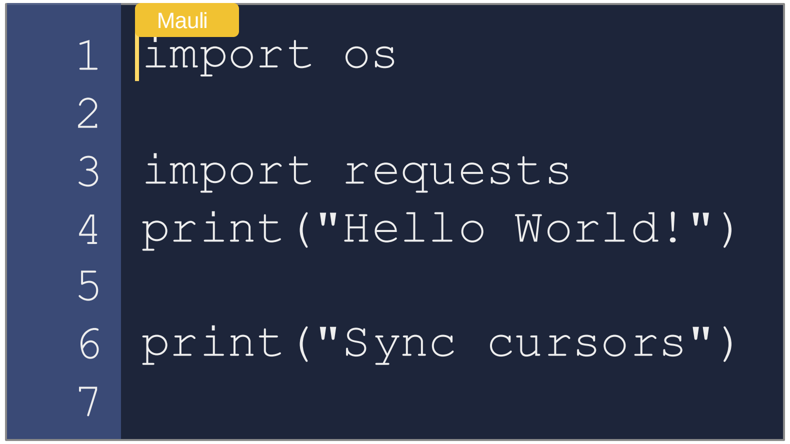
- Convey collaborator awareness cues (e.g. cursor location, cursor movement, content changes, etc.) non-visually with sound effects and speech.
- Design Criteria
    1. Support tightly-coupled collaboration
    2. Minimize the cognitive load on blind and visually impaired developers
    3. Maintain the agency of blind and visually impaired developers
    4. Reduce the burden on blind and visually impaired developers of driving the collaboration



```
import * as path from 'path';
import * as vscode from 'vscode';

const cats = {
    'Coding Cat': 'https://media.giphy.com/media/JIX9t2j0ZTN9S/giphy.gif',
    'Compiling Cat': 'https://media.giphy.com/media/mlvseq9yvZhba/giphy.gif',
                                                'Testing Cat': 'https://media.giphy.com/media/3oriO0OEd9QIDdllqo/giphy.gif'
};

export function activate(context: vscode.ExtensionContext) {
    context.subscriptions.push(
        vscode.commands.registerCommand('catCoding.start', () => {
            CatCodingPanel.createOrShow(context.extensionPath);
        })
    );

    context.subscriptions.push(
        vscode.commands.registerCommand('catCoding.doRefactor', () => {
            if (CatCodingPanel.currentPanel) {
                CatCodingPanel.currentPanel.doRefactor();
            }
        })
    );

    if (vscode.window.registerWebviewPanelSerializer) {
        // Make sure we register a serializer in activation event
        vscode.window.registerWebviewPanelSerializer(CatCodingPanel.viewType, {
            async deserializeWebviewPanel(webviewPanel: vscode.WebviewPanel, state: any) {
                console.log(`Got state: ${state}`);
                CatCodingPanel.revive(webviewPanel, context.extensionPath);
            }
        });
    }
}

/**
 * Manages cat coding webview panels
 */
```

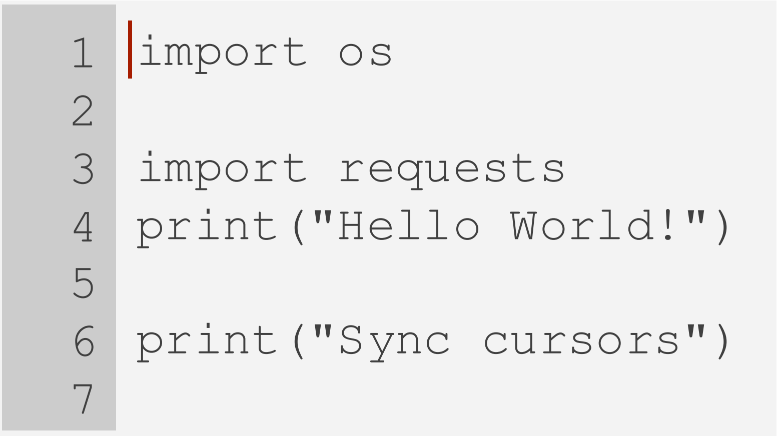
Are you using a screen reader to operate VS Code? (Certain features like word wrap are disabled when using a screen reader)

Cursor tethering keeps cursors in sync in Follow Mode



```
import os

import requests
print("Hello World!")

print("Sync cursors")

```

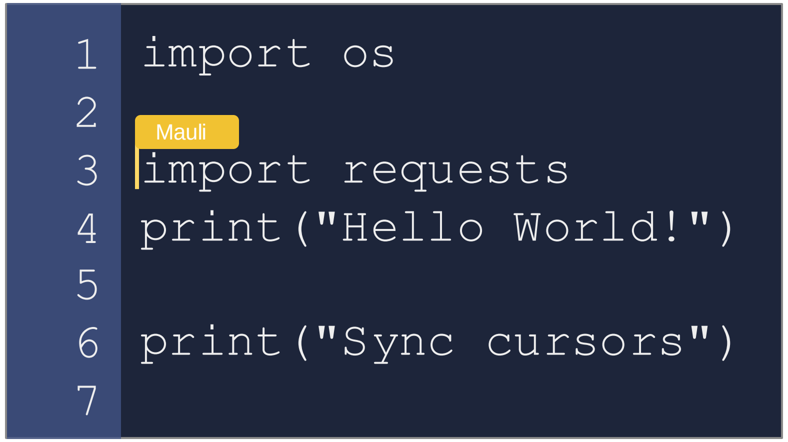
Follower's IDE



```
import os

import requests
print("Hello World!")

print("Sync cursors")

```

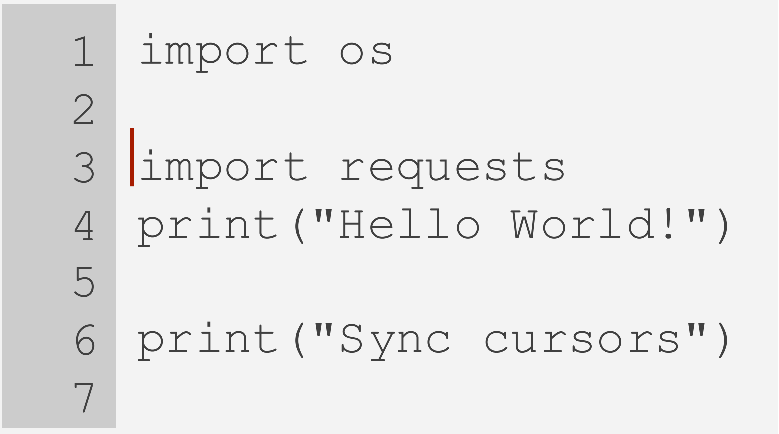
Driver's IDE

Cursor tethering keeps cursors in sync in Follow Mode



```
import os

import requests
print("Hello World!")

print("Sync cursors")

```

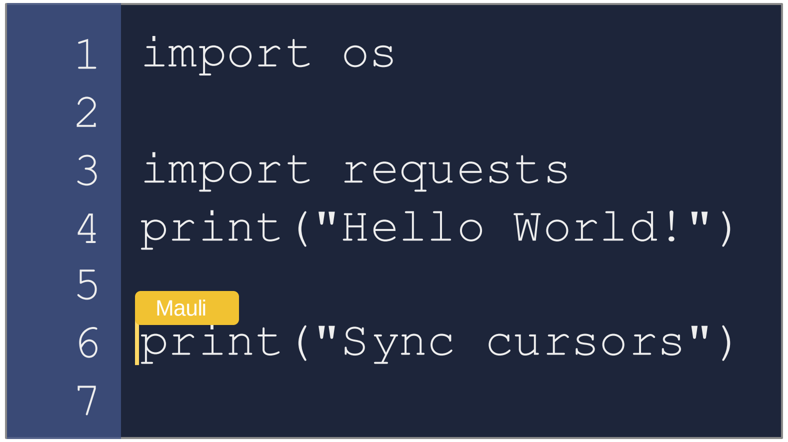
Follower's IDE



```
import os

import requests
print("Hello World!")

print("Sync cursors")

```

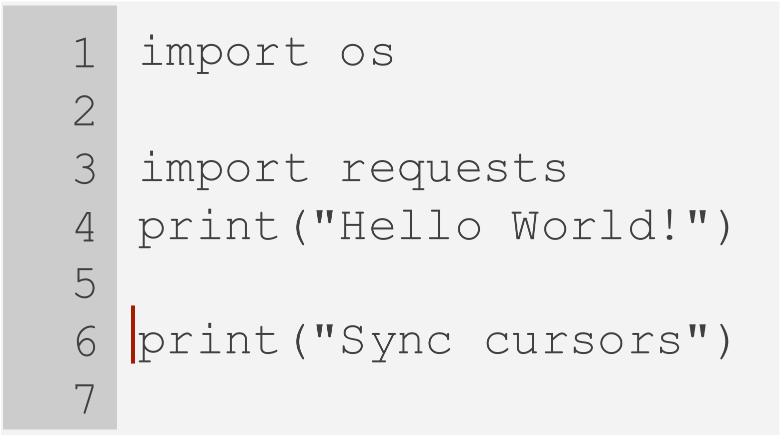
Driver's IDE

Cursor tethering keeps cursors in sync in Follow Mode



```
import os

import requests
print("Hello World!")

print("Sync cursors")

```

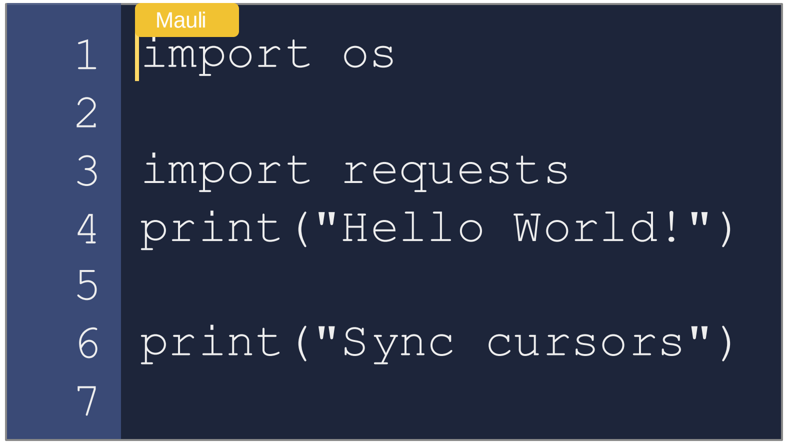
Follower's IDE



```
import os

import requests
print("Hello World!")

print("Sync cursors")

```

Driver's IDE

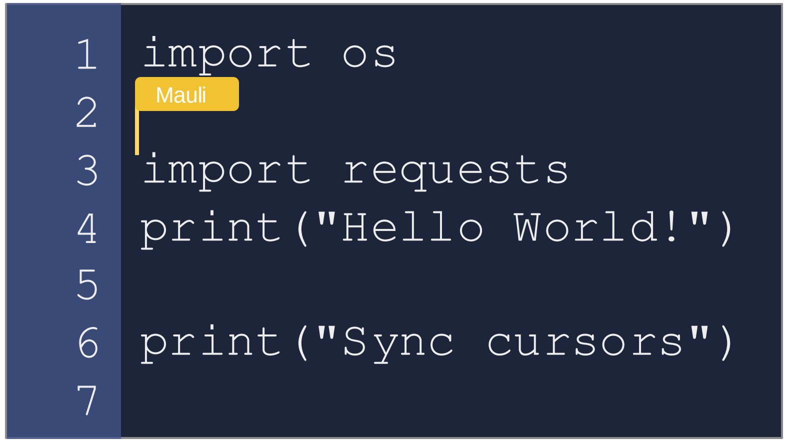
Sound effects and speech



```
1 import os
2
3 import requests
4 print("Hello World!")
5
6 print("Sync cursors")
7
```

Driver goes from line 1 to line 3 using arrows keys

Sound effects and speech

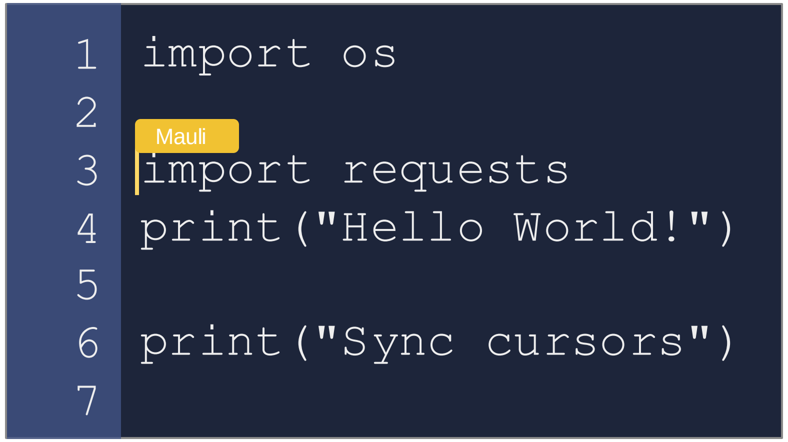


```
import os

import requests
print("Hello World!")

print("Sync cursors")

```

Follower's IDE

Keyboard Sound

Sound effects and speech

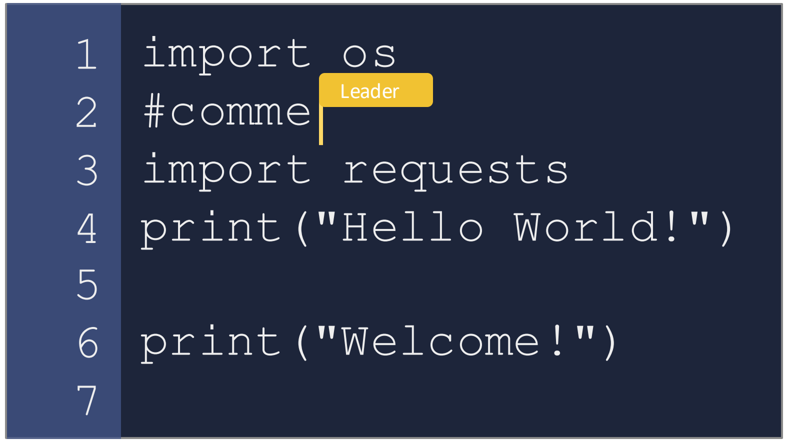


```
1  import os
2
3  import requests
4  print("Hello World!")
5
6  print("Sync cursors")
7
```

Follower's IDE

- Keyboard Sound
- Keyboard Sound
- Bend Sound
- "Line 3"

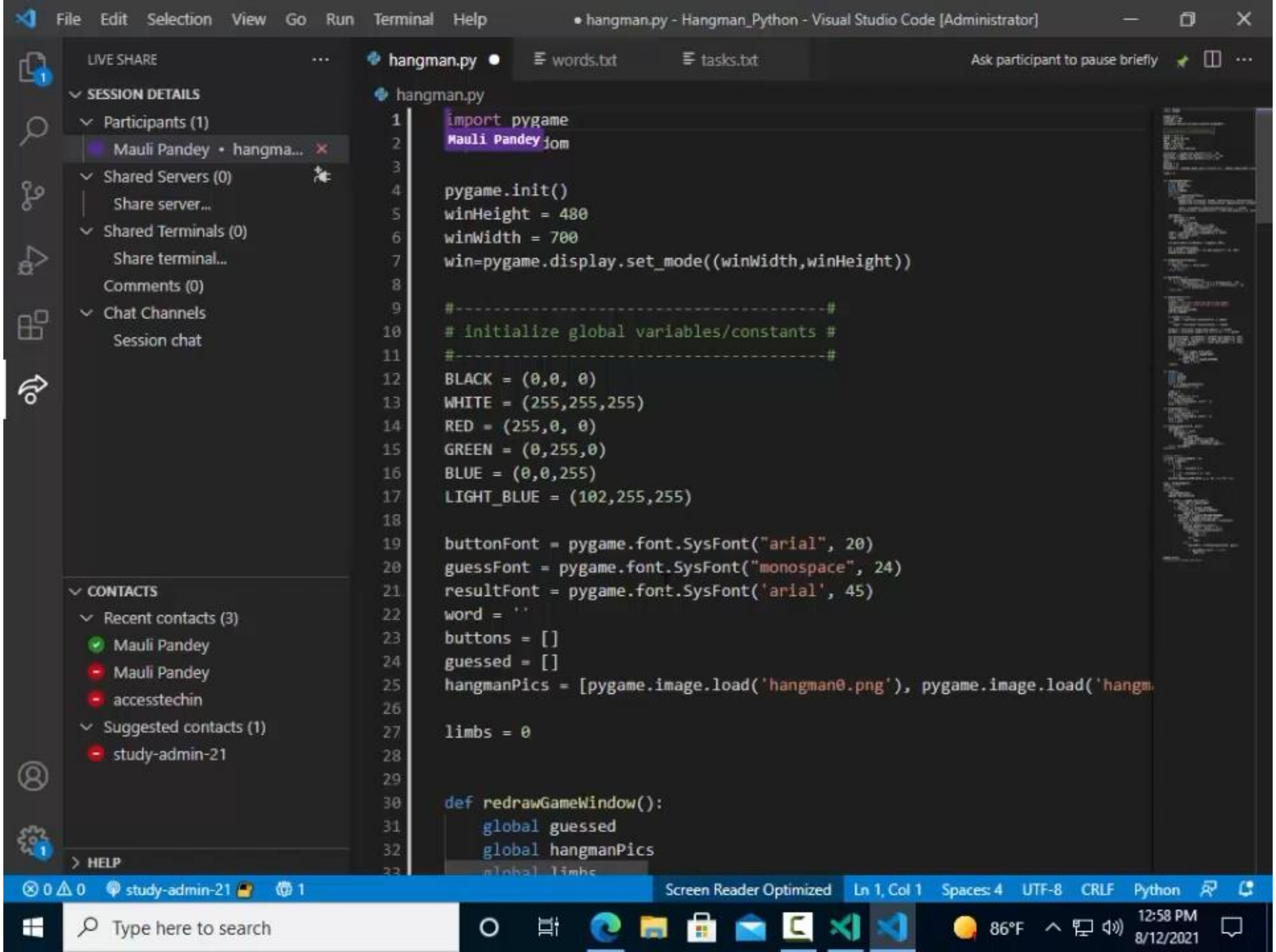
Edit awareness



```
import os
#comme

import requests
print("Hello World!")

print("Welcome!")

```

⌨ **Typing Sounds**
(*Follow Mode On*)

⌨ 💬 **Typing Sounds + Speech Announcement**
(Follow Mode Off)



```python
import pygame
                dom

pygame.init()
winHeight = 480
winWidth = 700
win=pygame.display.set_mode((winWidth,winHeight))

#---------------------------------------#
# initialize global variables/constants #
#---------------------------------------#
BLACK = (0,0, 0)
WHITE = (255,255,255)
RED = (255,0, 0)
GREEN = (0,255,0)
BLUE = (0,0,255)
LIGHT_BLUE = (102,255,255)

buttonFont = pygame.font.SysFont("arial", 20)
guessFont = pygame.font.SysFont("monospace", 24)
resultFont = pygame.font.SysFont('arial', 45)
word = ''
buttons = []
guessed = []
hangmanPics = [pygame.image.load('hangman0.png'), pygame.image.load('hangm

limbs = 0


def redrawGameWindow():
    global guessed
    global hangmanPics
    global limbs
```

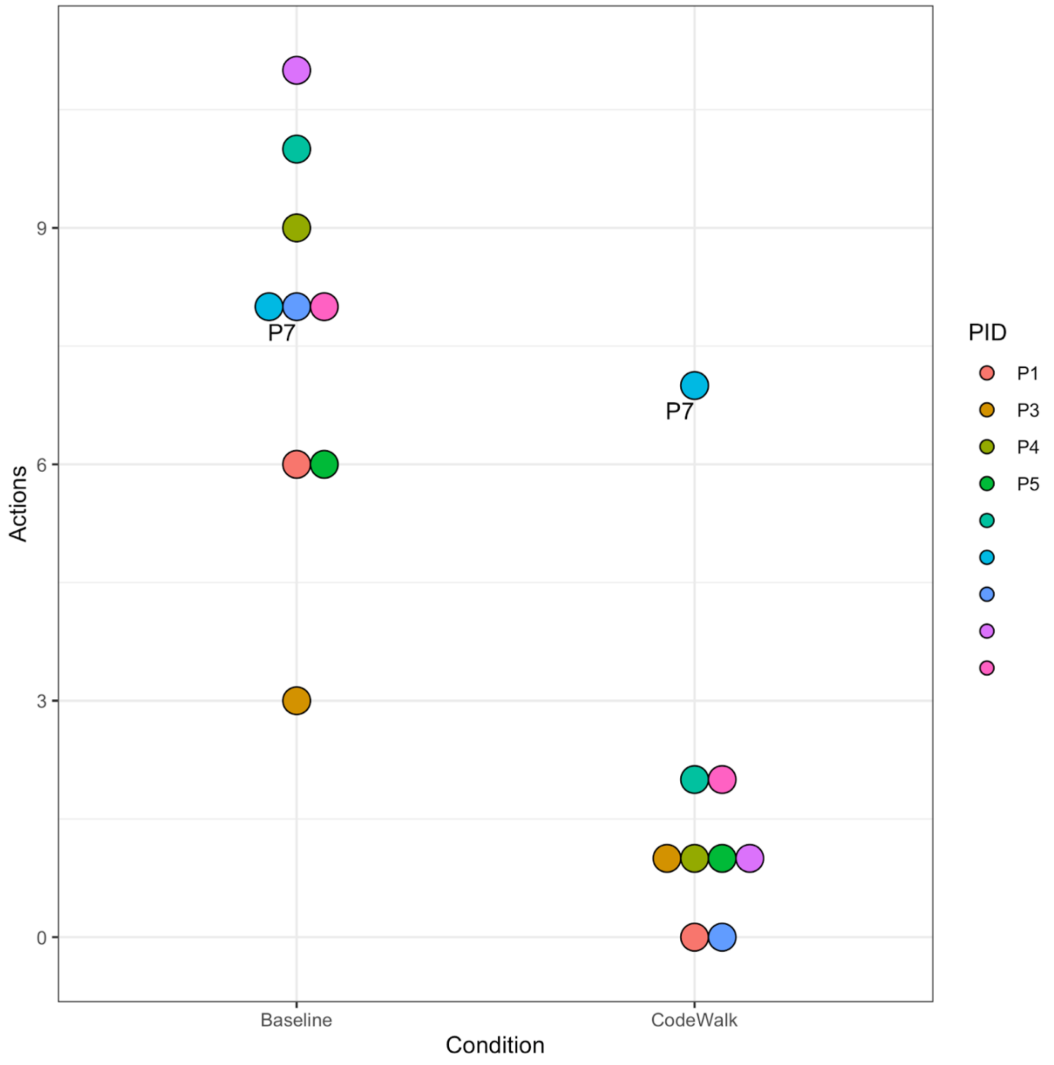
Evaluation

- We ran a within-subjects experiment with two conditions (Live Share and CodeWalk)
- Tasks included paired code editing, refactoring, and bug fixing
  - Mauli Pandey served as a sighted confederate for each blind or visually impaired participant developer
- Likert scale questionnaire after each condition
- Informal follow-up interview to understand participants' experience and to gather feedback

Results

- Codewalk significantly reduced the number of sync attempts during collaboration

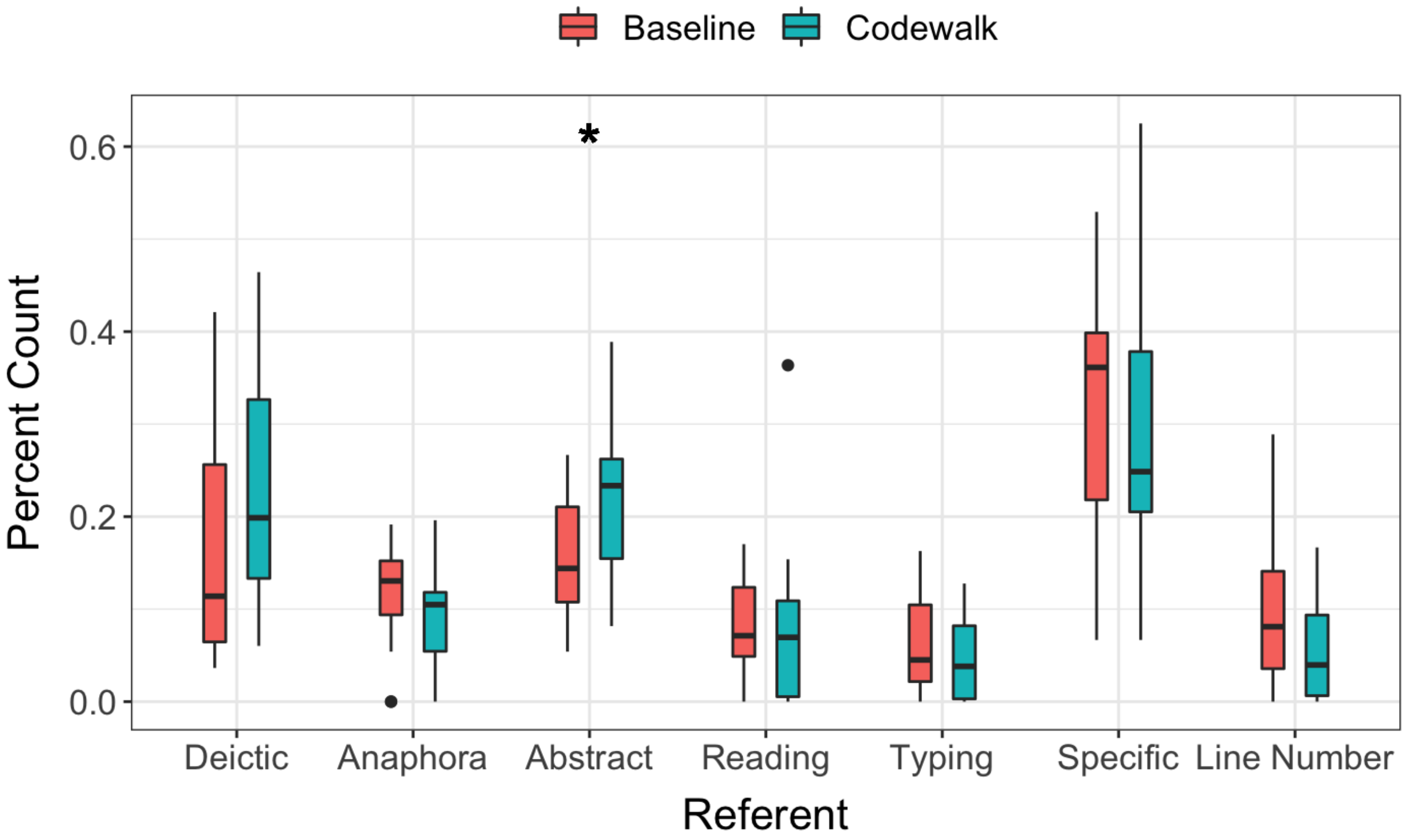
- Codewalk improved shared awareness between collaborators



Results

Participants referred to code locations in many different ways.

Abstract referents were spoken more often with CodeWalk.



Perception of Collaboration Experience

(all better in CodeWalk condition, except S9, S12)

S1	I was keenly aware of everything in my environment.
S2	I was conscious of what is going on around me.
S3	I was aware of what my teammate did and how it happened.
S4	I was aware that my teammate is aware of my actions.
S5	I am aware of how well we performed together in the team.
S6	I felt like my teammate and I were on the same page most of the time
S7	I could tell what my teammate was thinking about/looking at/talking about most of the time.
S8	I felt like we shared common subgoals as we worked on the task.
S9	My teammate communicated clearly during the task.
S10	I communicated clearly with my teammate during this task.
S11	It was fun to work with my teammate on this task.
S12	My teammate worked effectively with me to accomplish the task.

Results

> “So, the difference [between CodeWalk and Live Share] I think was just more [...] a sense of not being lost and a sense of not worrying that I’m not following you... Because I could just snap to wherever you were, I wasn’t worried about wandering off [...] It was just kind of more comfortable in a sense of awareness.

PARTICIPANT 4