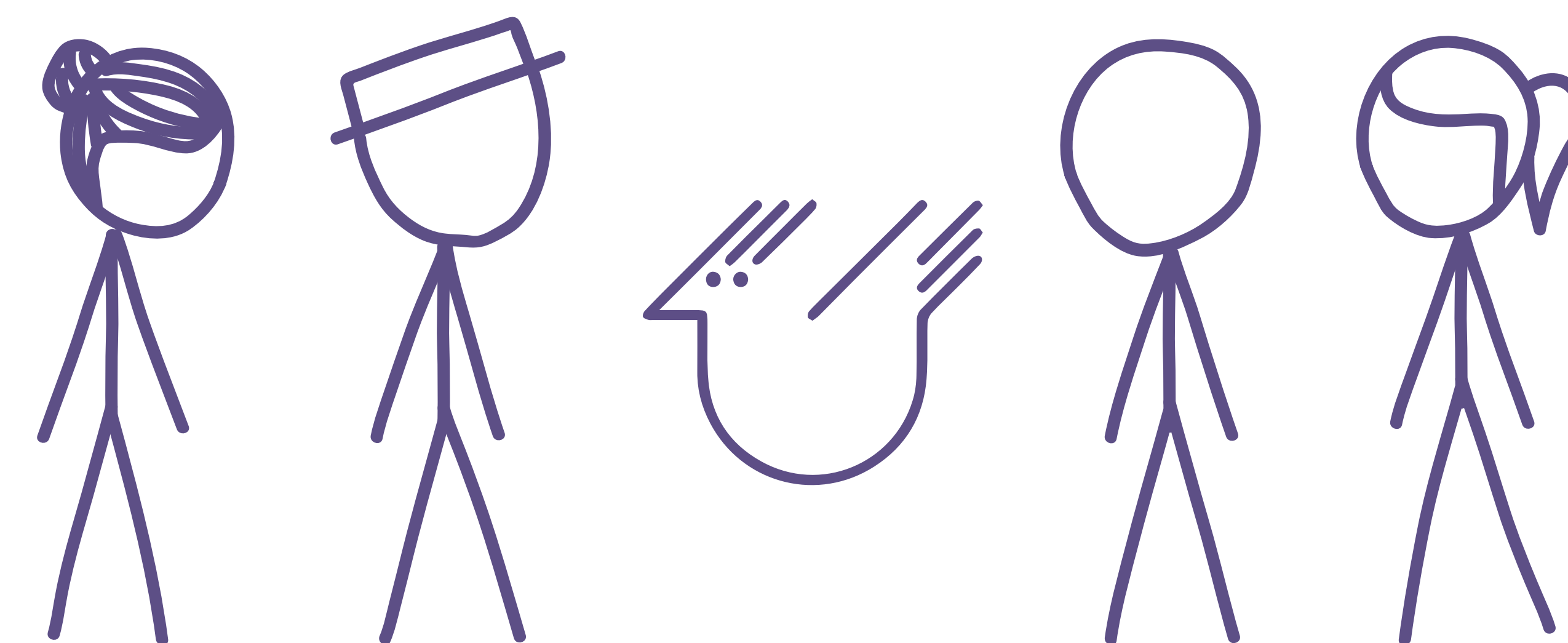


How Novices Perceive Interactive Theorem Provers



```
1 | module Util where
2 |
3 | swap : a × b → b × a
4 | swap (a, b) = (b , a)
5 |
```

```
1 | module Util where
2 |
3 | swap : a × b → b × a
4 | swap (a, b) = (b , a)
5 |
```

Could not parse the left-hand side swap (a, b)

Problematic expression: (a, b)

Operators used in the grammar:

None

when scope checking the left-hand side

swap (a, b) in the definition of swap

→ █

```
1 | module Util where
2 |
3 | swap : a × b → b × a
4 | swap (a, b) = (b , a)
5 |
```

1 attempt to understand the message

Could not parse the left-hand side swap (a, b)

Problematic expression: (a, b)

Operators used in the grammar:

None

when scope checking the left-hand side

swap (a, b) in the definition of swap

→



```
1 | module Util where
2 |
3 | swap : a × b → b × a
4 | swap (a, b) = (b , a)
5 |
```

1 attempt to understand the message
scope checking? operators? None?!

Could not parse the left-hand side swap (a, b)

Problematic expression: (a, b)

Operators used in the grammar:

None

when scope checking the left-hand side

swap (a, b) in the definition of swap

→



```
1 | module Util where
2 |
3 | swap : a × b → b × a
4 | swap (a, b) = (b , a)
5 |
```

1 attempt to understand the message
scope checking? operators? None?!

2 ask the internet

Could not parse the left-hand side swap (a, b)

Problematic expression: (a, b)

Operators used in the grammar:

None

when scope checking the left-hand side

swap (a, b) in the definition of swap

→



```
1 | module Util where
2 |
3 | swap : a × b → b × a
4 | swap (a, b) = (b , a)
5 |
```

1 attempt to understand the message
scope checking? operators? None?!

2 ask the internet

 no results found

Could not parse the left-hand side swap (a, b)

Problematic expression: (a, b)

Operators used in the grammar:

None

when scope checking the left-hand side

swap (a, b) in the definition of swap

→

```
1 | module Util where
2 |
3 | swap : a × b → b × a
4 | swap (a, b) = (b , a)
5 |
```

1 attempt to understand the message
scope checking? operators? None?!

2 ask the internet

 no results found

3 stare at the code

Could not parse the left-hand side swap (a, b)

Problematic expression: (a, b)

Operators used in the grammar:

None

when scope checking the left-hand side

swap (a, b) in the definition of swap

→ █

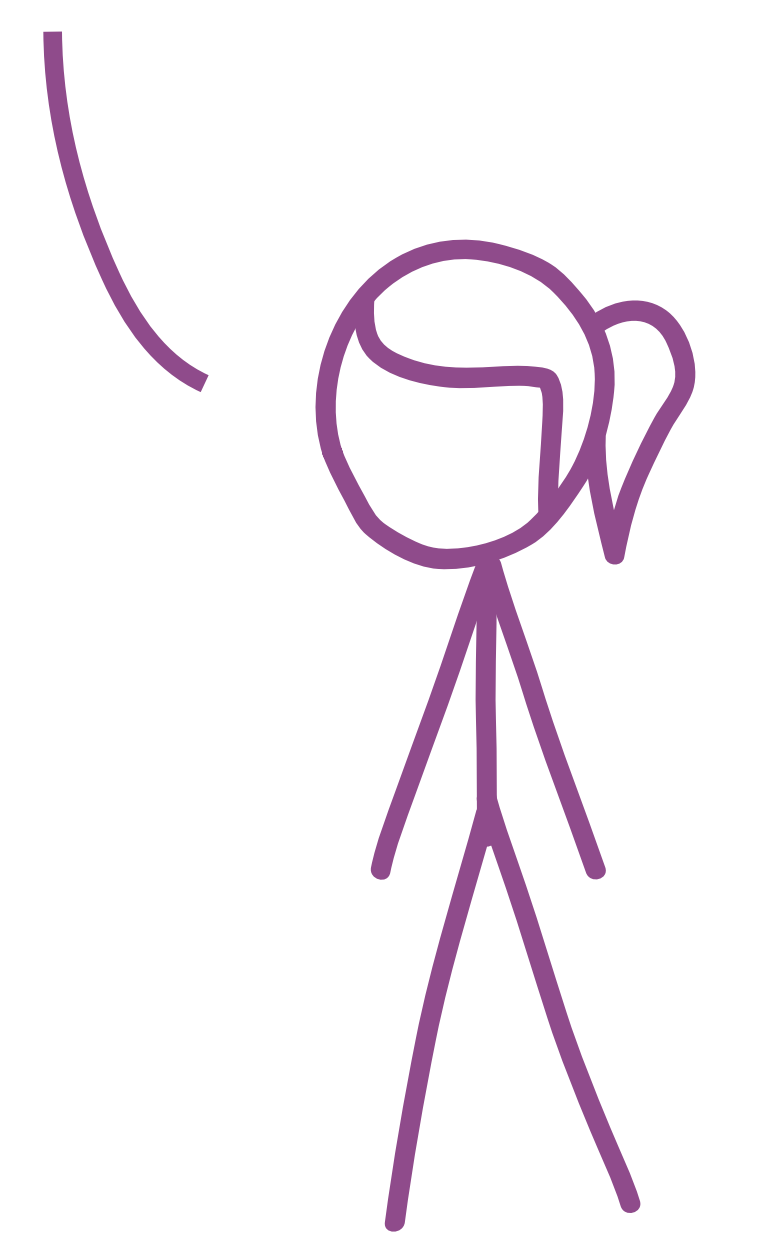

```
1 | module Util where
2 |
3 | swap : a × b → b × a
4 | swap (a, b) = (b , a)
5 |
```

1 attempt to understand the message
scope checking? operators? None?!

2 ask the internet

 no results found

3 stare at the code
ummm... where's the
syntax highlighting?



Could not parse the left-hand side swap (a, b)


Problematic expression: (a, b)

Operators used in the grammar:

None

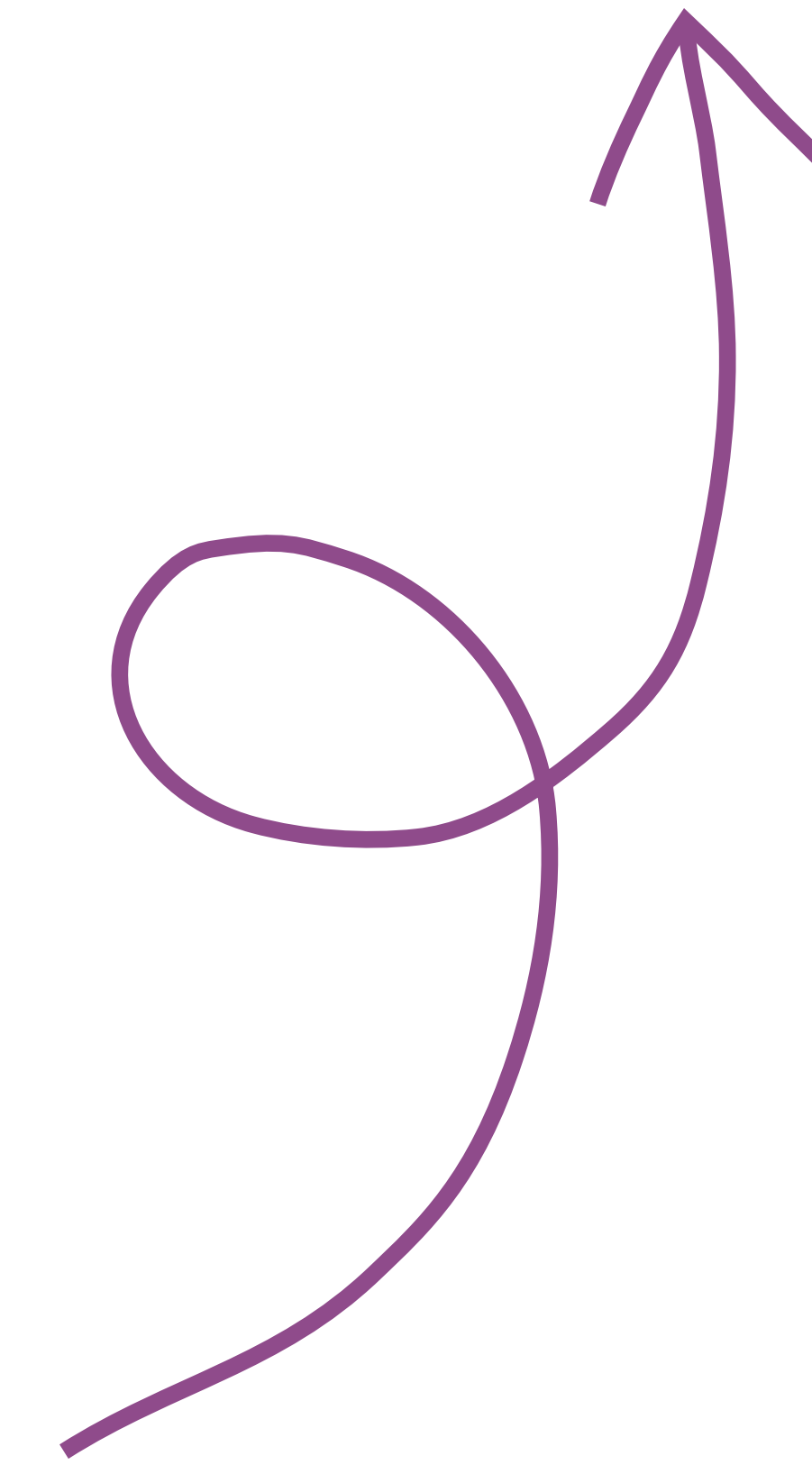
when scope checking the left-hand side

swap (a, b) in the definition of swap

→ 

```
1 | module Util where
2 |
3 | swap : a × b → b × a
4 | swap (a , b) = (b , a)
5 |           ↑
```

“there is a space missing on line 2, column 8”



Could not parse the left-hand side swap (a, b)

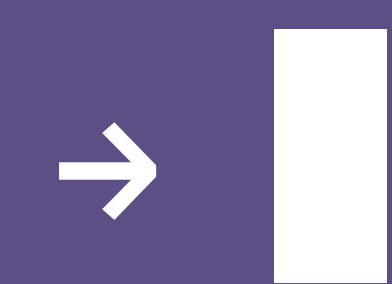
Problematic expression: (a, b)

Operators used in the grammar:

None

when scope checking the left-hand side

swap (a, b) in the definition of swap



Goal:

long-term: identify which aspects of interactive theorem provers to improve with respect to usability

Goal:

long-term: identify which aspects of interactive theorem provers to improve with respect to usability

this talk: identify which obstacles novices face when learning to use an interactive theorem prover

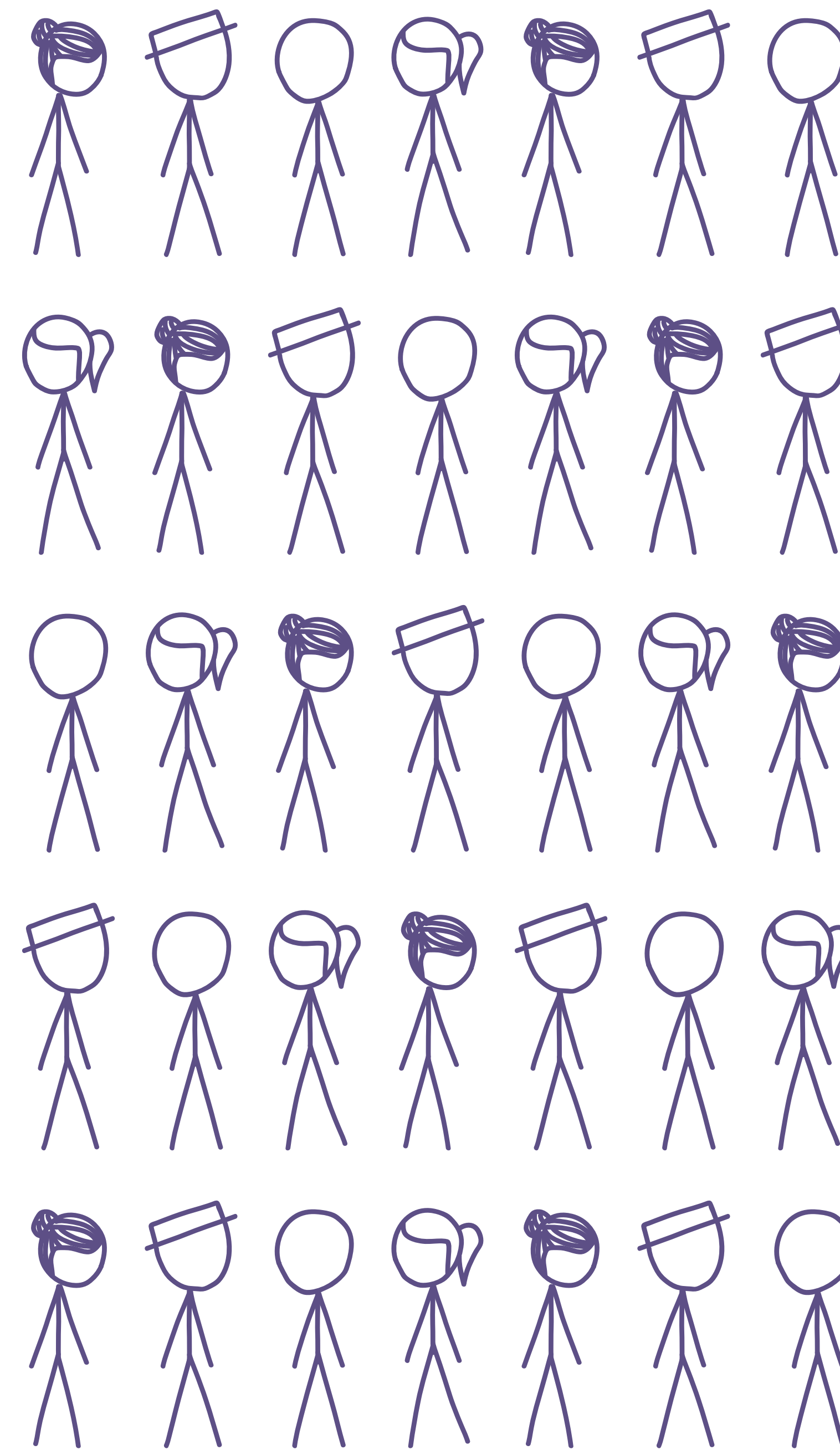
This Talk:

1. Study Setup
2. Identified Obstacles
3. Observations
4. The Future

STUDY SETUP

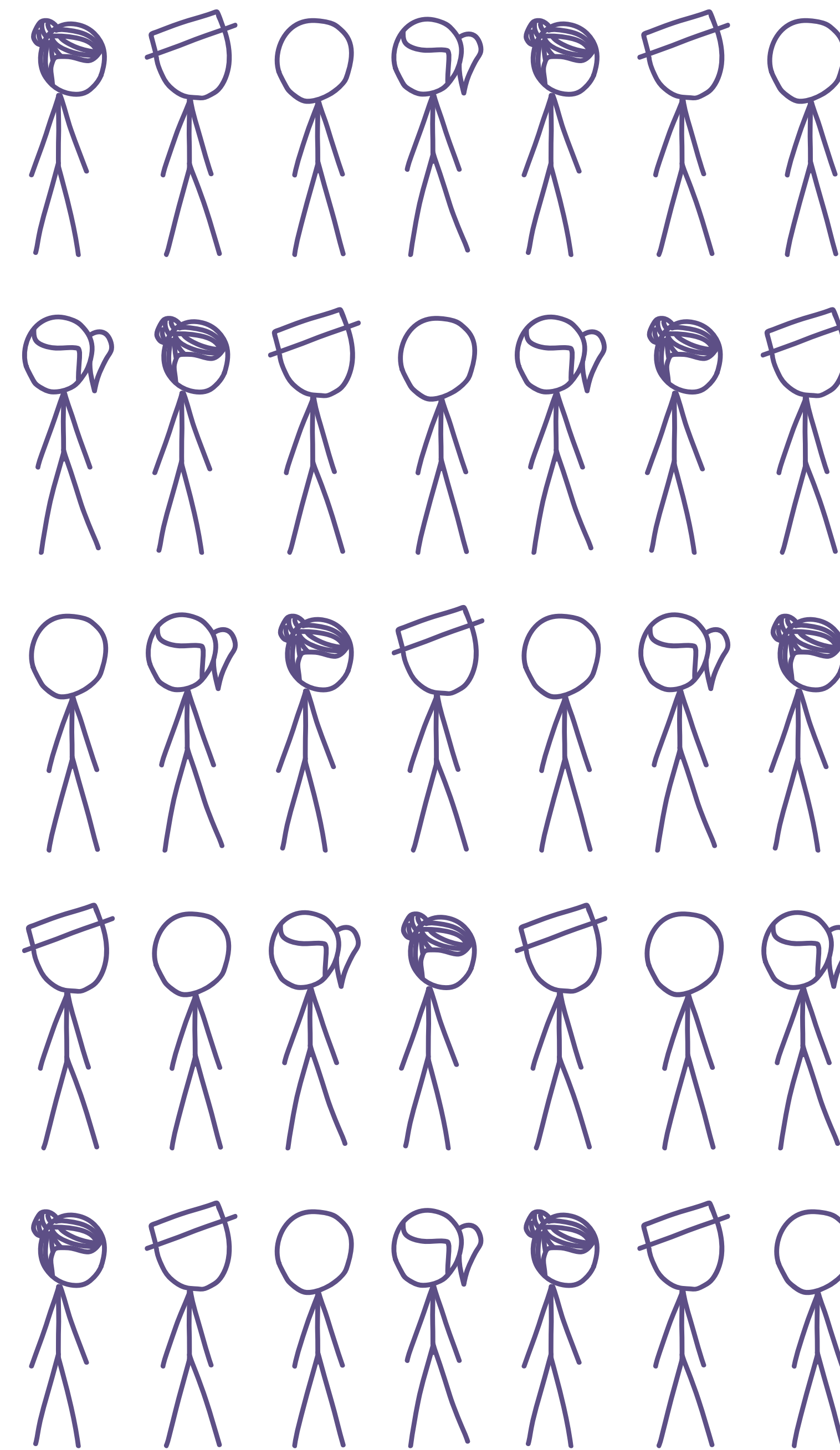
35 bachelor students

taking a course on
Functional Programming



35 bachelor students

taking a course on
Functional Programming



Course Material

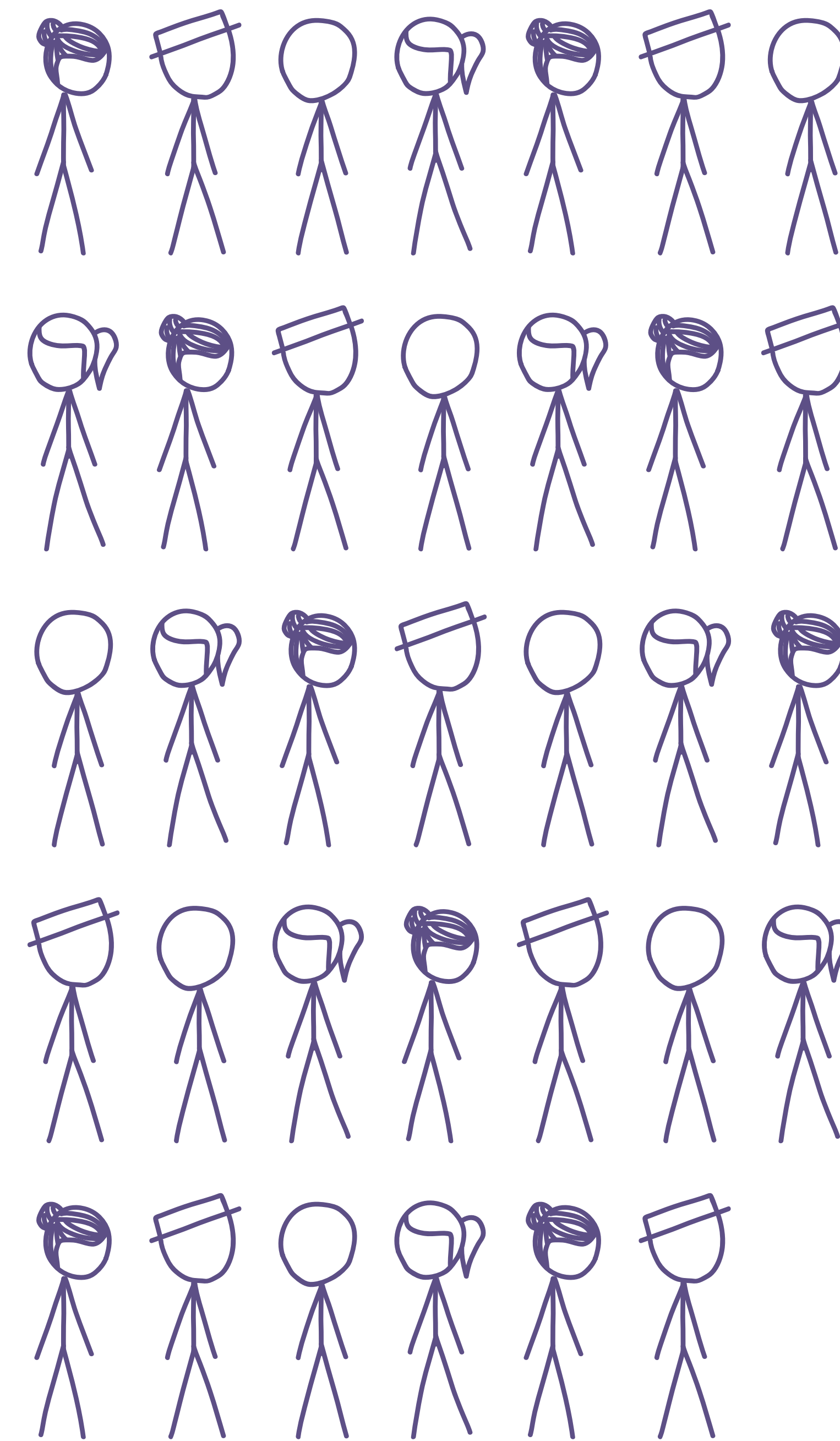
1. basics of functional programming in Haskell
2. introduction to Agda

- interactive development
- Curry-Howard correspondence
- indexed data types
- dependent pattern matching
- formal proofs using the identity type and equational reasoning

(using the agda-mode extension in VS Code)

35 bachelor students

taking a course on
Functional Programming



(one participant)



Course Material

1. basics of functional programming in Haskell
2. introduction to Agda

- interactive development
- Curry-Howard correspondence
- indexed data types
- dependent pattern matching
- formal proofs using the identity type and equational reasoning

(using the agda-mode extension in VS Code)

+ more knowledge

https://qualtrics.com/agda-novices

List up to five obstacles you encountered when learning to use Agda:

1:

2:

3:

4:

5:

- online
- open-ended
- five short, free-text fields
- accepts up to five answers

Using qualitative research techniques:

- coding
- diagramming
- memo-writing

Coding

code

“Unicode in the code was
hard to get used to.”

syntax

“The Agda plugin in my [VSC] was often
crashing and I had to restart it.”

tooling

Coding

code

“Unicode in the code was
hard to get used to.”

syntax

“The Agda plugin in my [VSC] was often
crashing and I had to restart it.”

tooling

“Syntax highlighting completely disappears
if there is some mistake in the code which
makes it harder to find the mistake.”

tooling

Coding

code

sub-code

“Unicode in the code was
hard to get used to.”

syntax

unicode

“The Agda plugin in my [VSC] was often
crashing and I had to restart it.”

tooling

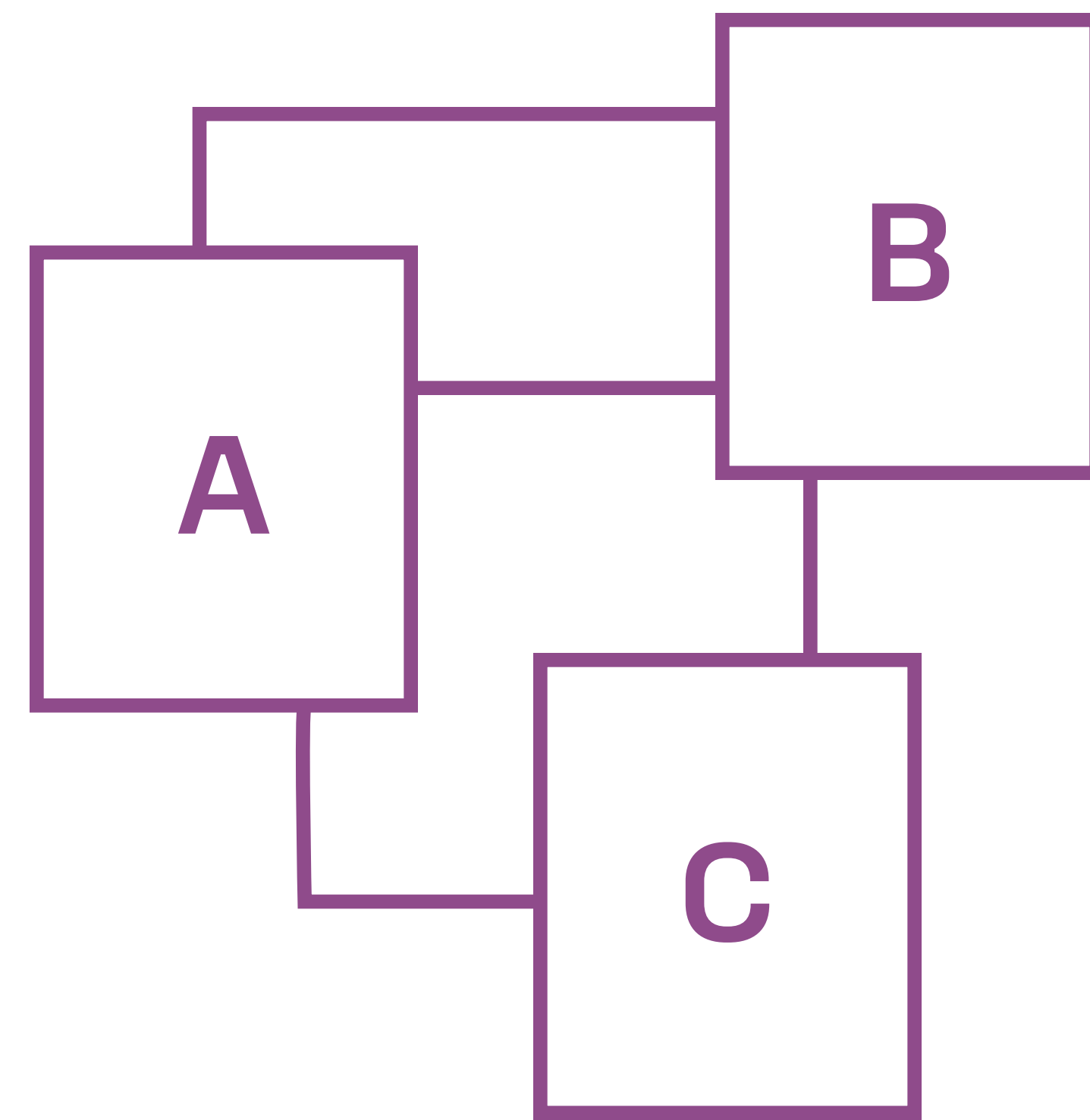
unintended,
buggy behaviour

“Syntax highlighting completely disappears
if there is some mistake in the code which
makes it harder to find the mistake.”

tooling

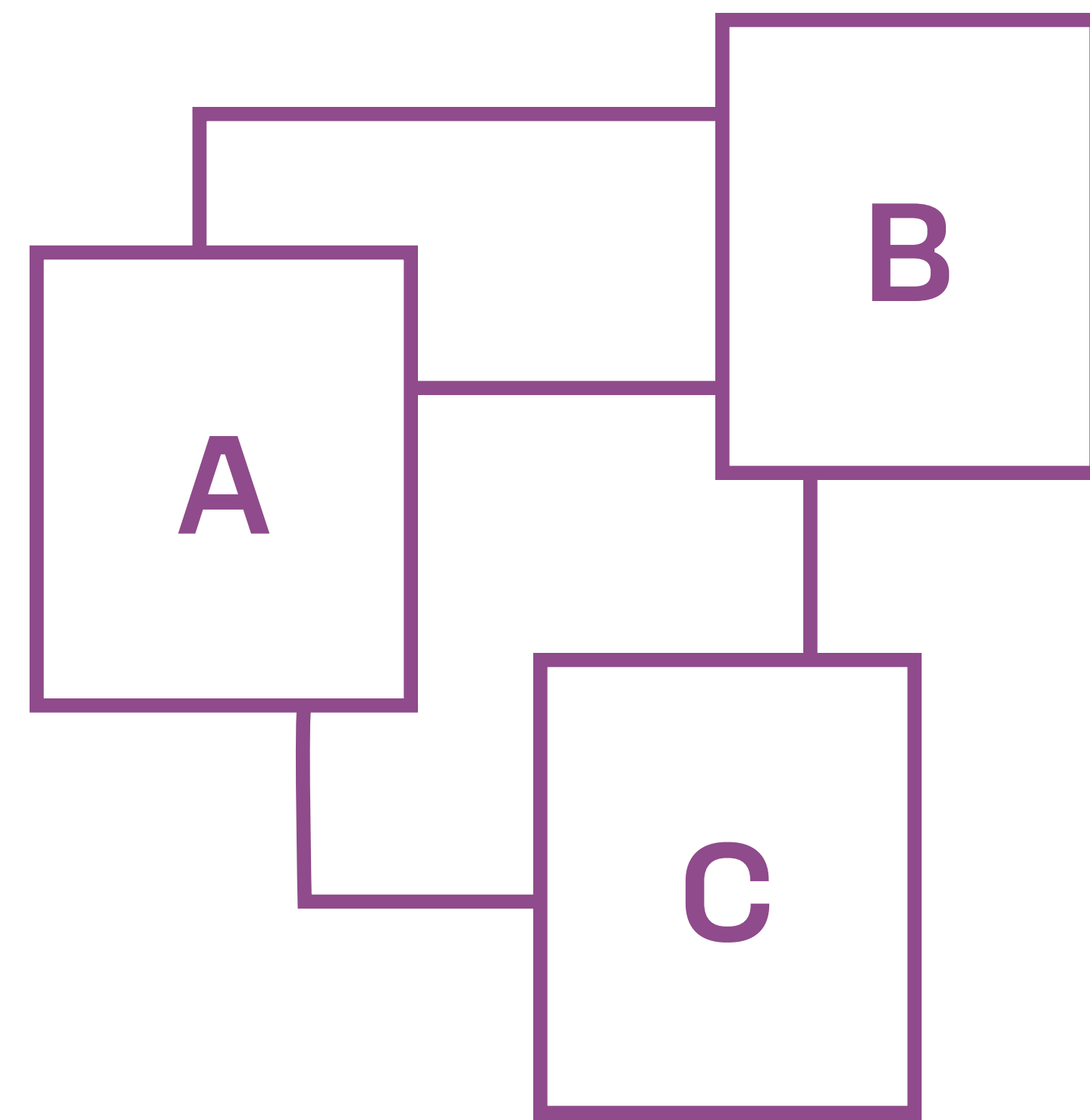
impractical
design

Diagramming



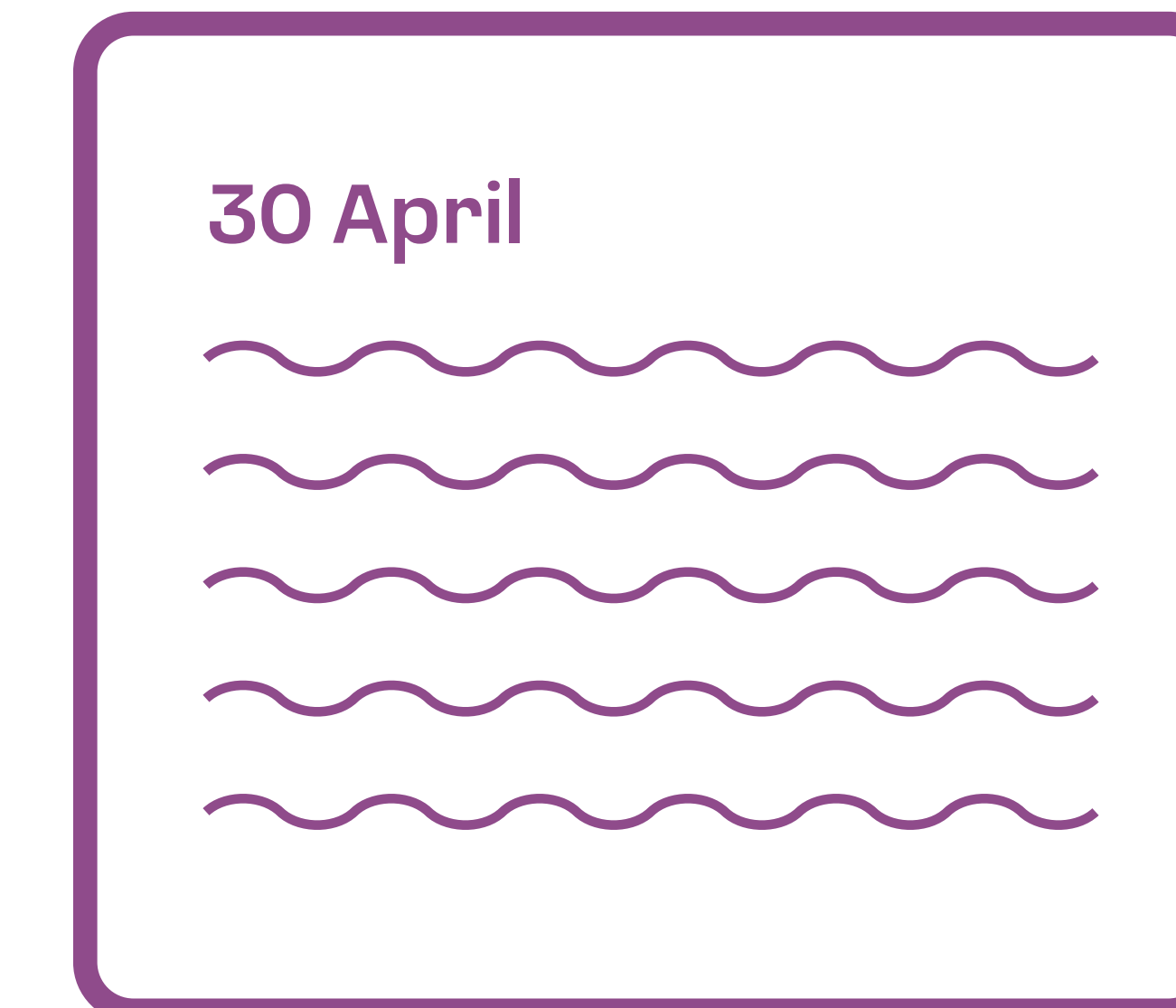
- identify related categories of obstacles
- understand relationships between obstacles

Diagramming



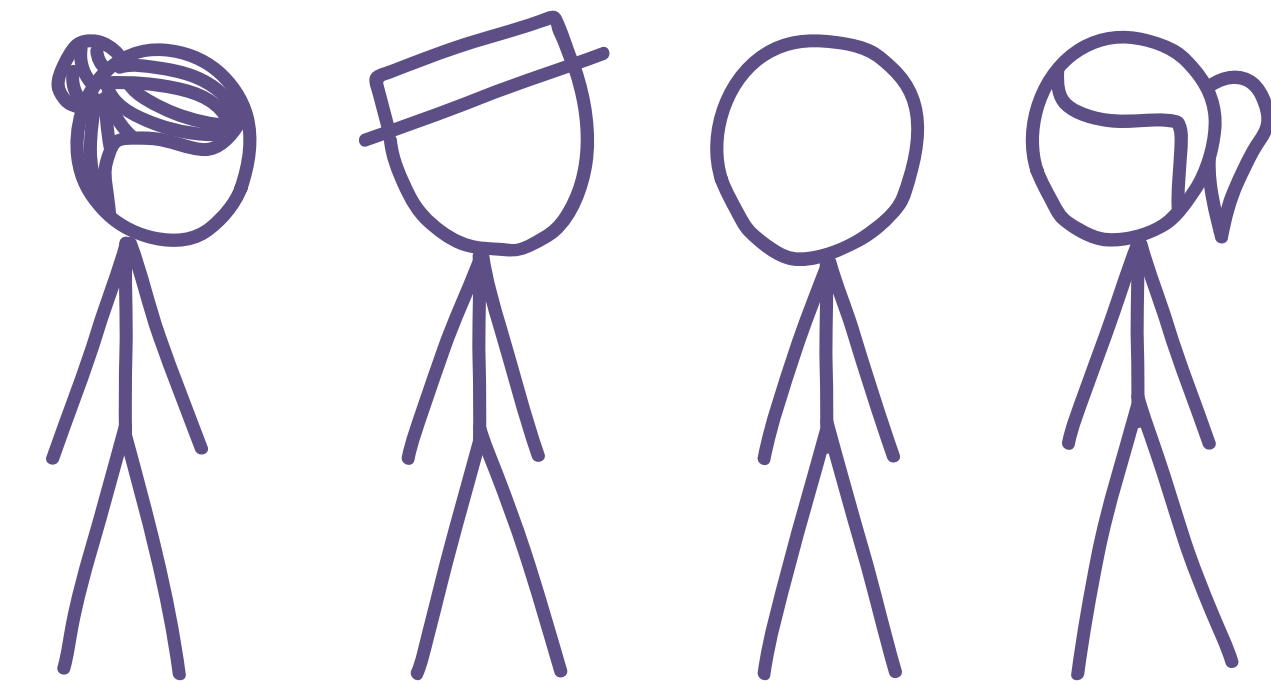
- identify related categories of obstacles
- understand relationships between obstacles

Memo-writing



notes of “ideas about codes and their relationships as they strike the analyst”¹

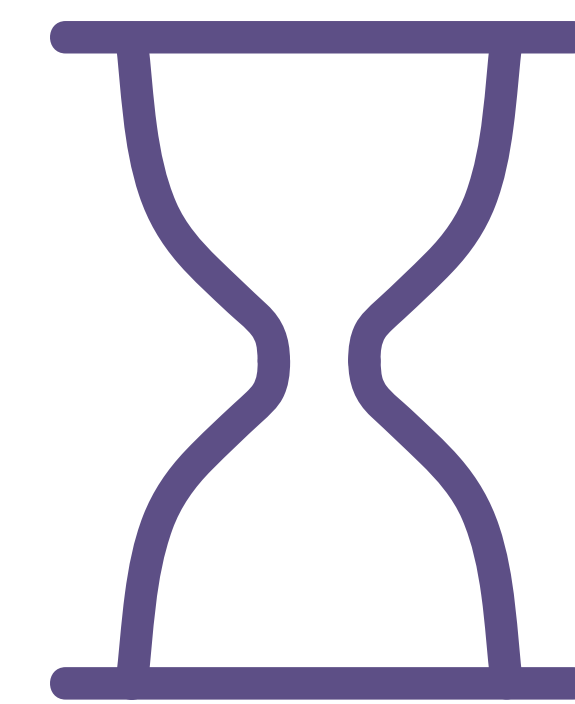
Barney G. Glaser. 1978. “Theoretical sensitivity: Advances in the methodology of grounded theory.”



a homogeneous set of students



learning to use only one ITP



in a short span of time

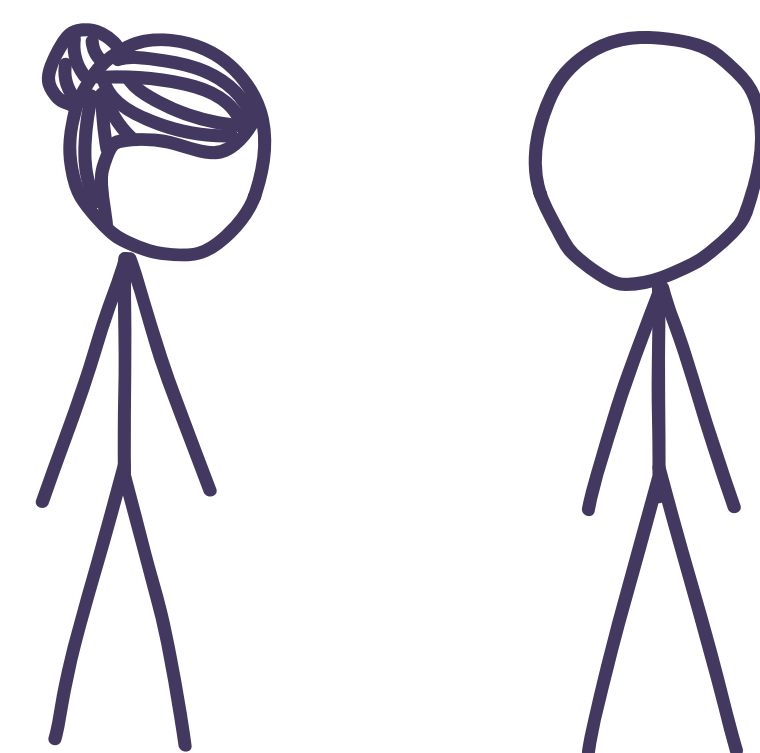
OBSTACLES

on the level of
Theory,
Implementation,
and Applicability in the Real World

Unfamiliar Concepts

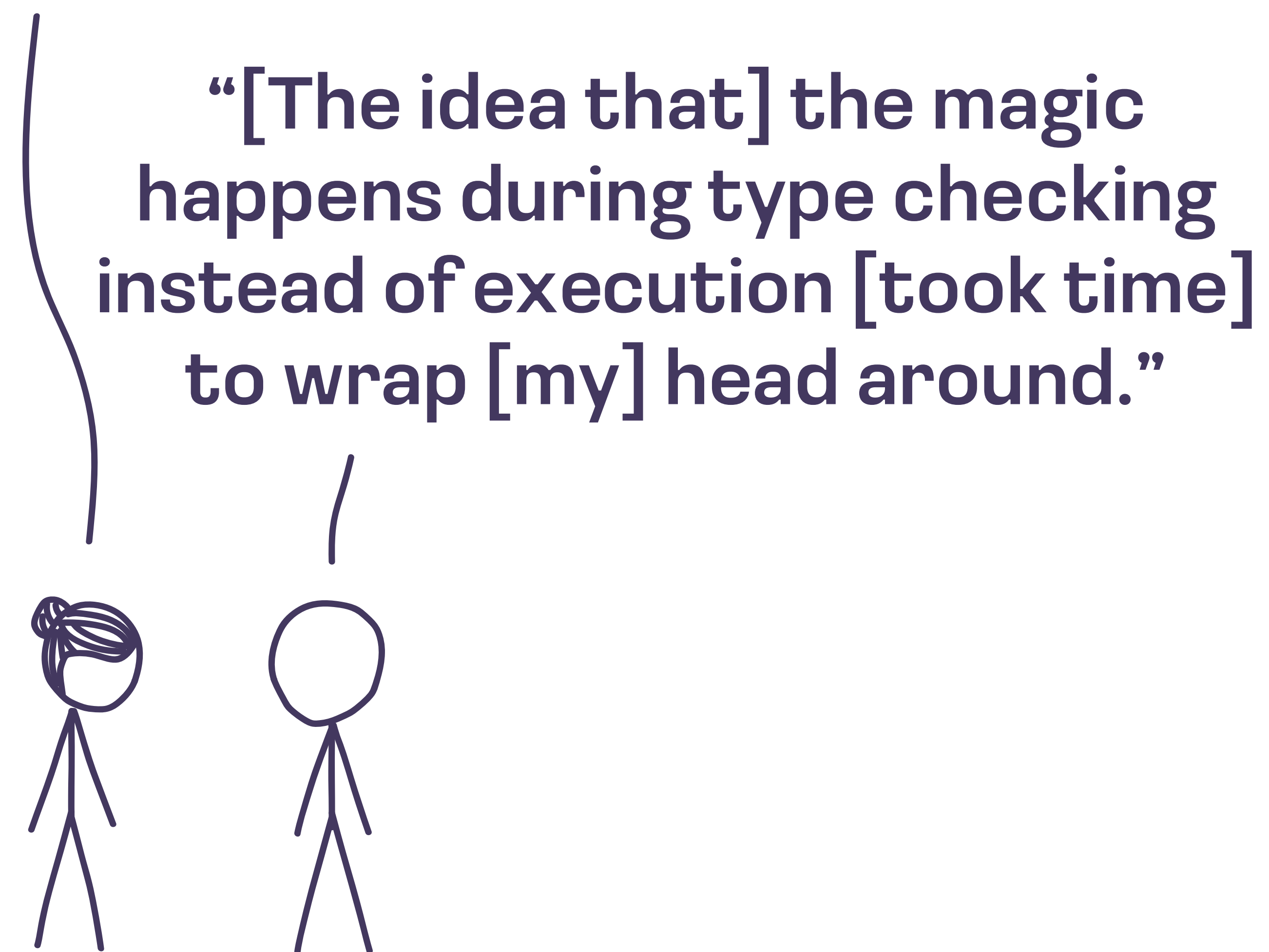
“Dependent types were not intuitive.”

“[The idea that] the magic happens during type checking instead of execution [took time] to wrap [my] head around.”



Unfamiliar Concepts

“Dependent types were not intuitive.”



Complex Theory

“The way you need to think about your program [is] much more abstract.”



“Weird” Design

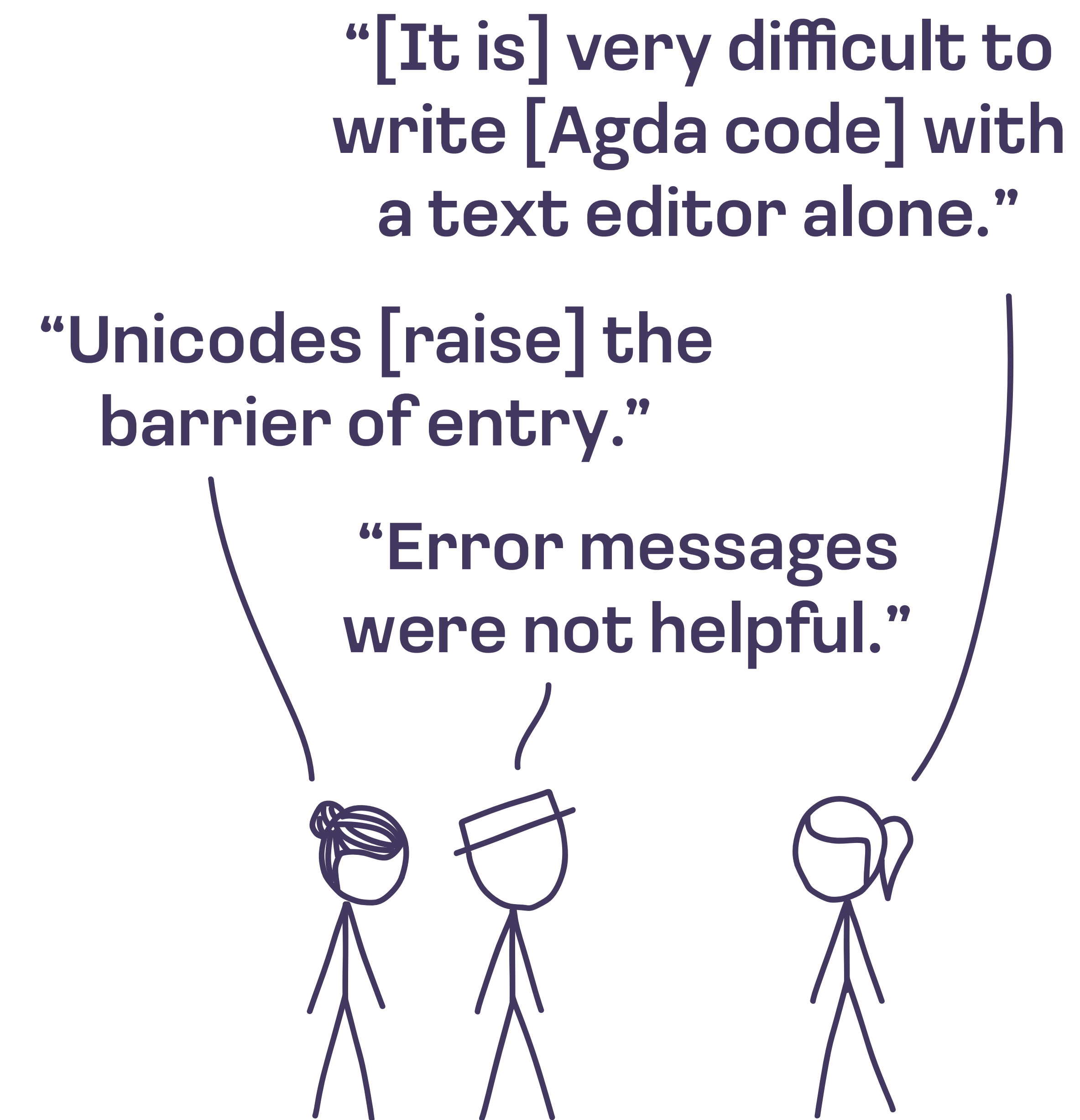
Examples include:

- Unicode characters,
- non-standard spacing rules
- error messages requiring theoretical knowledge and experience to be helpful
- abstractions requiring an understanding of the implementation

“Weird” Design

Examples include:

- Unicode characters,
- non-standard spacing rules
- error messages requiring theoretical knowledge and experience to be helpful
- abstractions requiring an understanding of the implementation



Inadequate Ecosystem

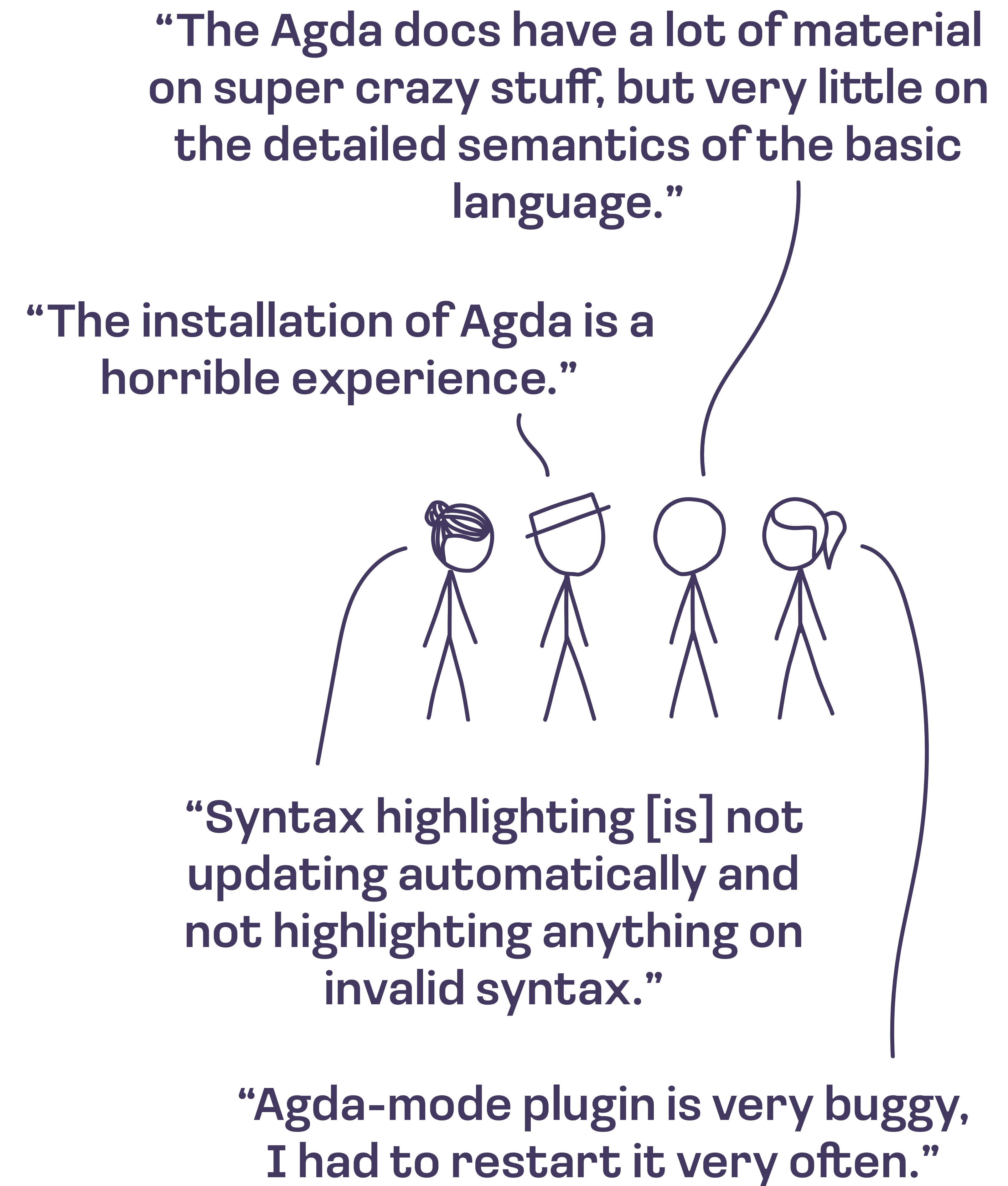
The ecosystem that supports program-writing in Agda is:

- incomplete,
- buggy,
- inconvenient,
- and not accessible to novices.

Inadequate Ecosystem

The ecosystem that supports program-writing in Agda is:

- incomplete,
- buggy,
- inconvenient,
- and not accessible to novices.



Perceived Irrelevance

“[Agda] might be a bit too theoretical.”

“It's hard to imagine writing software with Agda.”



OBSERVATIONS

attempt to understand the message

scope checking? operators? None?!

1

Many obstacles are a result of the high coupling between Agda's underlying theory and its design.

Util.agda ×

```
1 | module Util where
2 |
3 | swap : a × b → b × a
4 | swap (a, b) = (b , a)
5 |
```

Could not parse the left-hand side swap (a, b)

Problematic expression: (a, b)

Operators used in the grammar:

None

when scope checking the left-hand side

swap (a, b) in the definition of swap

→

ask the internet

 no results found

1

Many obstacles are a result of the high coupling between Agda's underlying theory and its design.

2

Agda's ecosystem has very little supporting infrastructure for novices.

Util.agda ×

```
1 | module Util where
2 |
3 | swap : a × b → b × a
4 | swap (a, b) = (b , a)
5 |
```

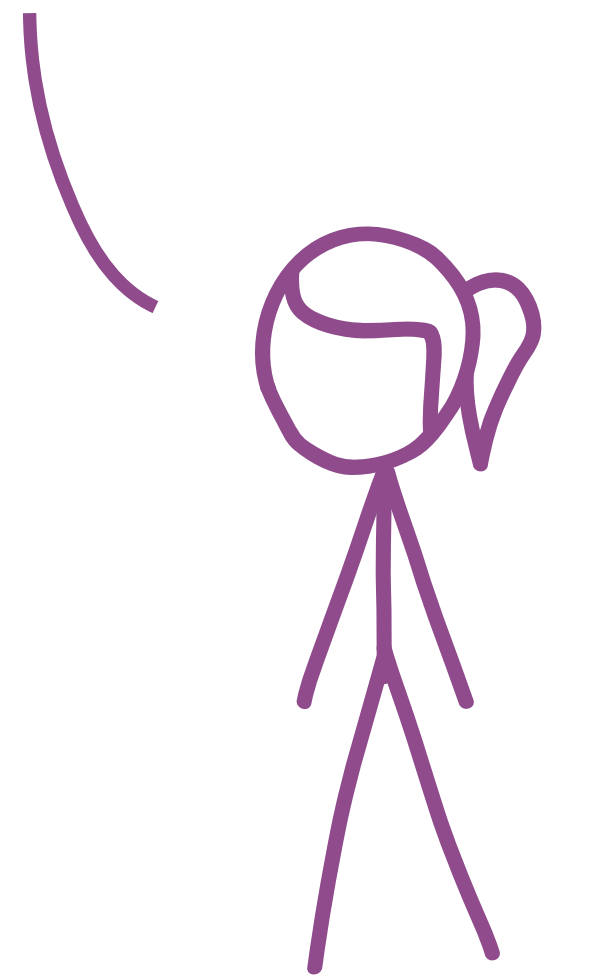
```
Could not parse the left-hand side swap (a, b)
Problematic expression: (a, b)
Operators used in the grammar:
  None
when scope checking the left-hand side
swap (a, b) in the definition of swap
```

→

- 1 Many obstacles are a result of the high coupling between Agda's underlying theory and its design.
- 2 Agda's ecosystem has very little supporting infrastructure for novices.
- 3 Agda's design makes it dependent on a custom (and not user friendly) development environment.

stare at the code
ummm... where's the
syntax highlighting?

```
Util.agda x
1 | module Util where
2 |
3 | swap : a × b → b × a
4 | swap (a, b) = (b , a)
5 |
```

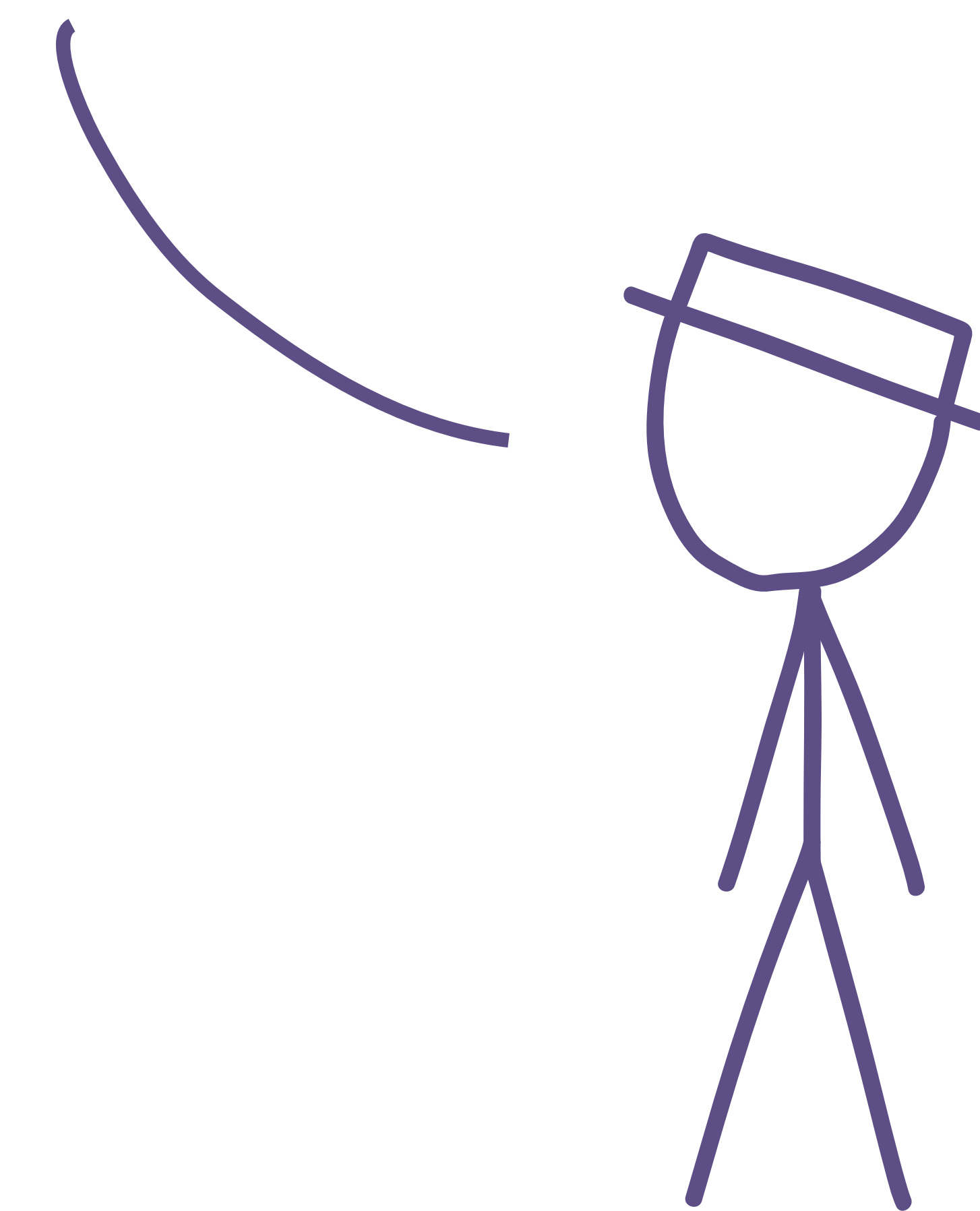


```
Could not parse the left-hand side swap (a, b)
Problematic expression: (a, b)
Operators used in the grammar:
  None
when scope checking the left-hand side
swap (a, b) in the definition of swap
→ |
```

- 1 Many obstacles are a result of the high coupling between Agda's underlying theory and its design.
- 2 Agda's ecosystem has very little supporting infrastructure for novices.
- 3 Agda's design makes it dependent on a custom (and not user friendly) development environment.

- 1 Many obstacles are a result of the high coupling between Agda's underlying theory and its design.
- 2 Agda's ecosystem has very little supporting infrastructure for novices.
- 3 Agda's design makes it dependent on a custom (and not user friendly) development environment.

IT LOOKS LIKE WE NEED TO FIND A WAY TO MAKE THE INFRASTRUCTURE MORE ACCESSIBLE AND STURDY!



THE FUTURE

1

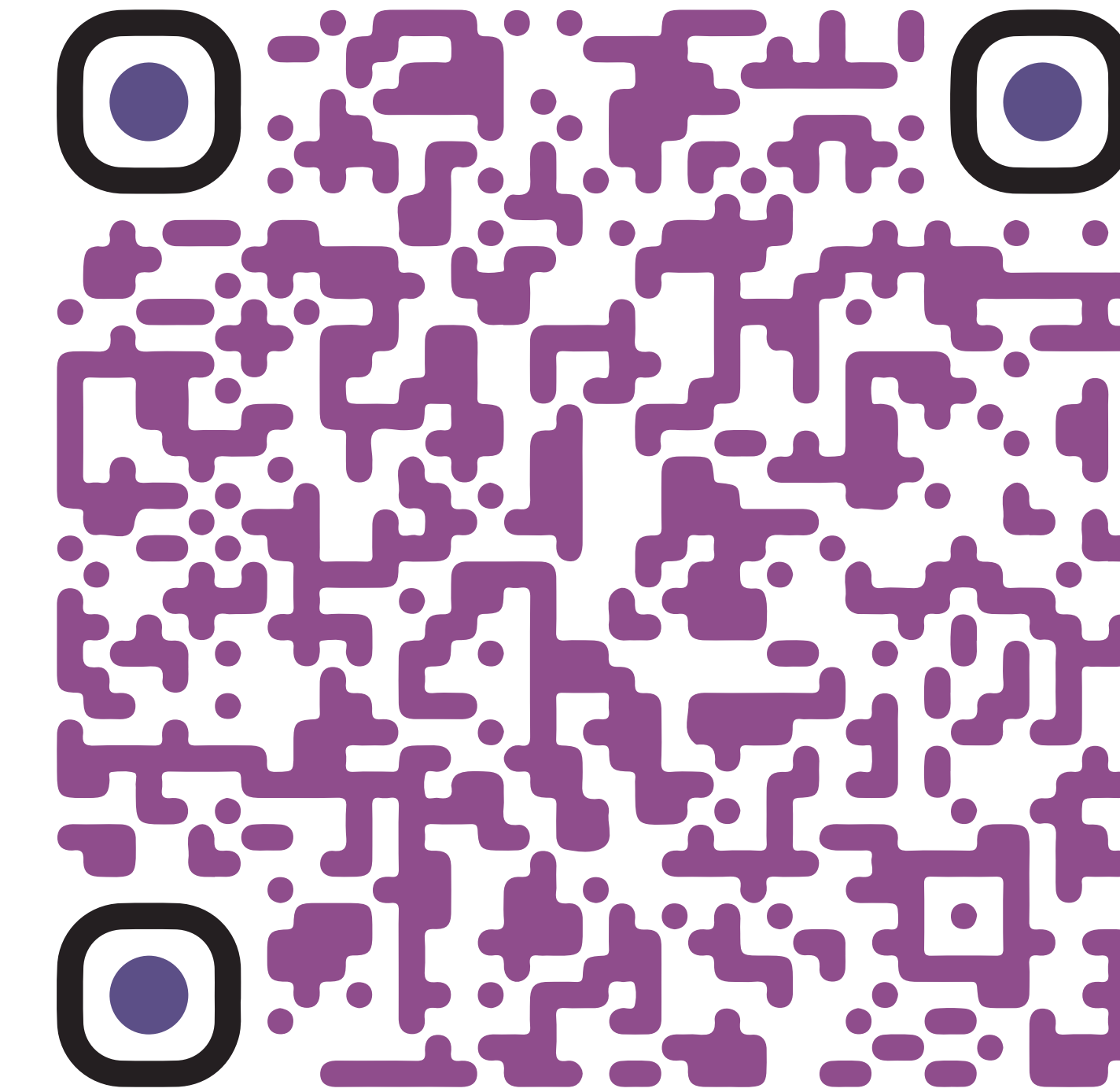
Focus on a more varied user base and a wider selection of interactive theorem provers.

1

Focus on a more varied user base and a wider selection of interactive theorem provers.



Sign up to participate in a study on how we use interactive theorem provers!

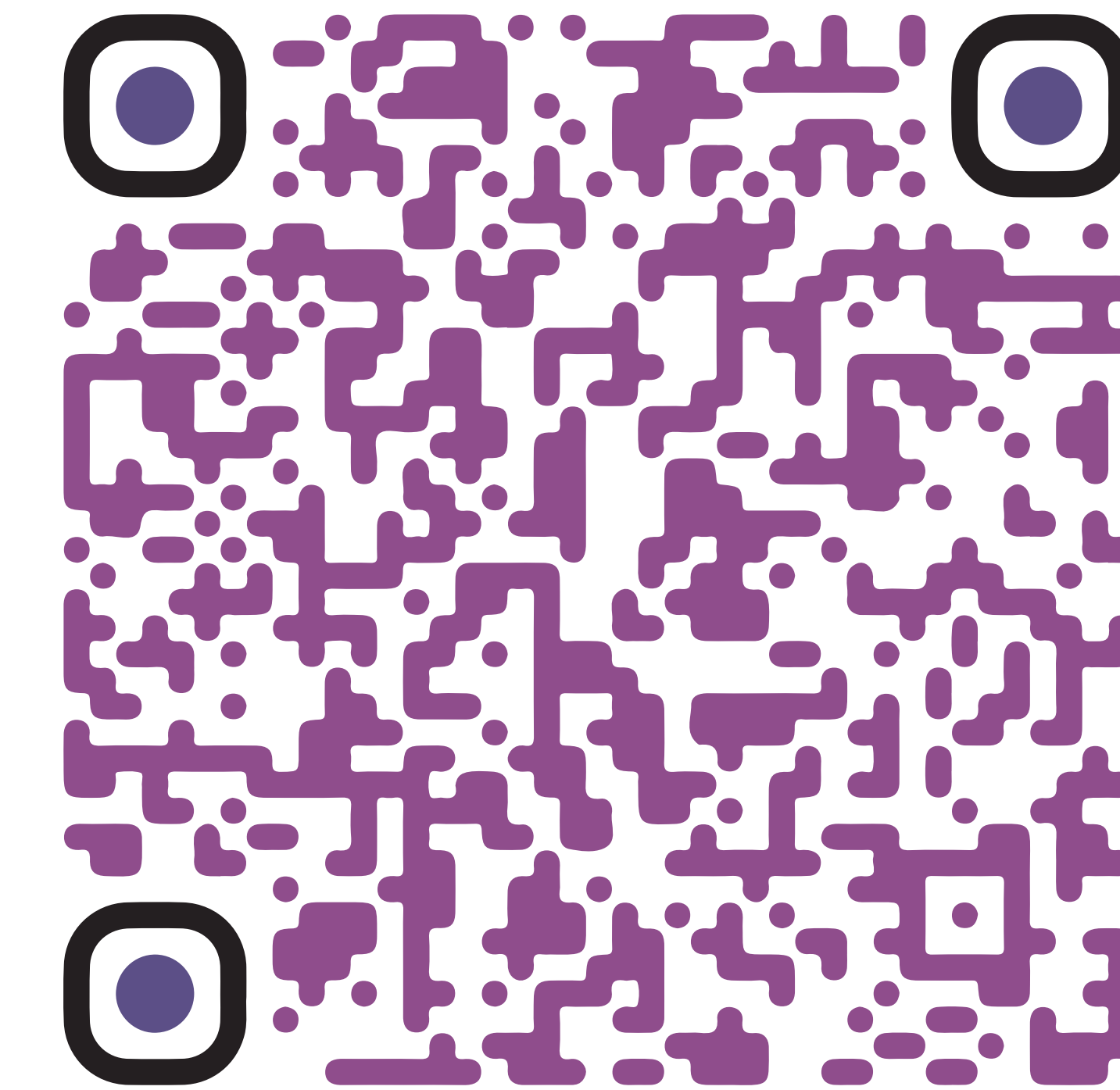


1

Focus on a more varied user base and a wider selection of interactive theorem provers.



Sign up to participate in a study on how we use interactive theorem provers!



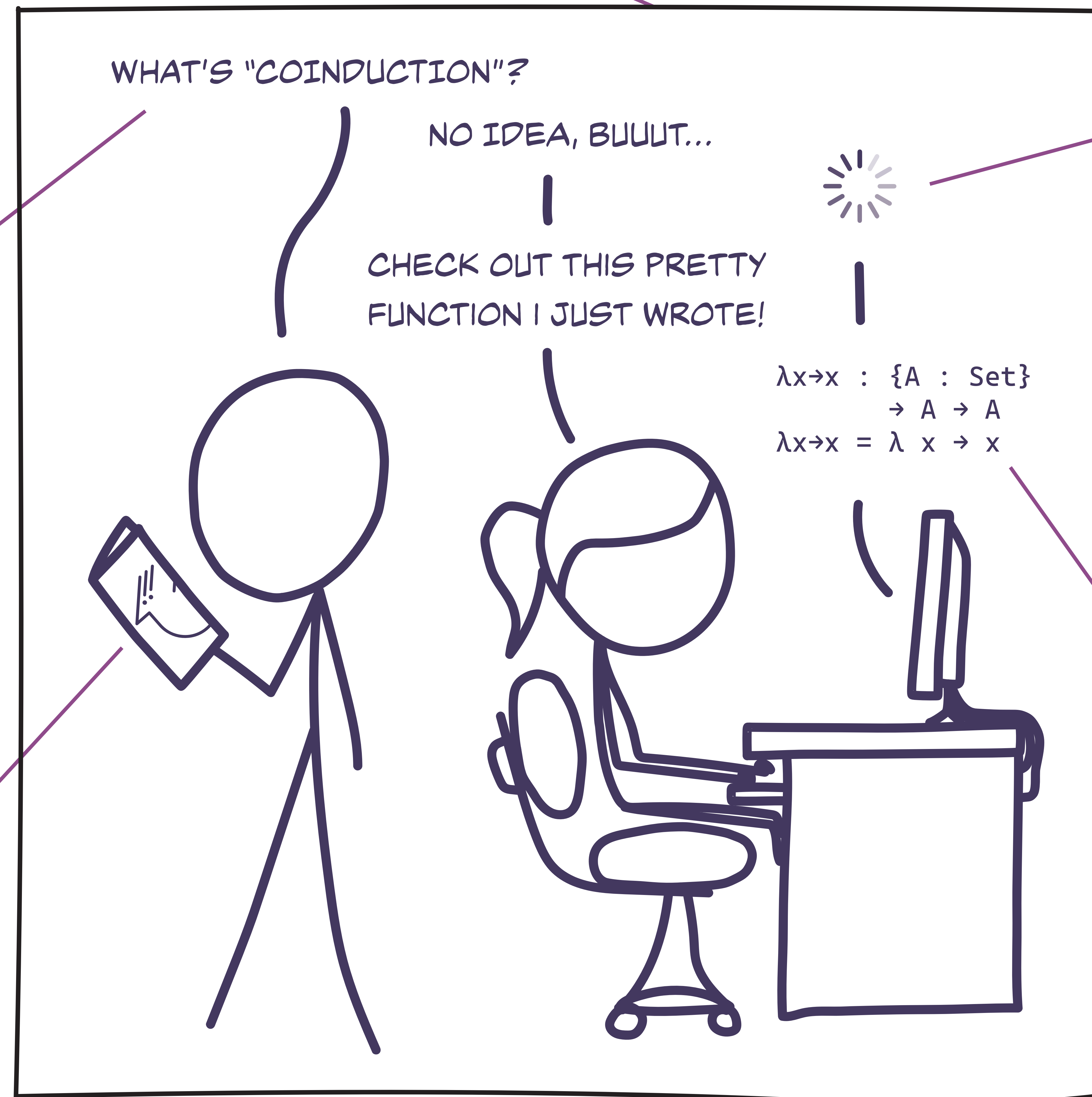
2

Find solutions to the technical challenges encountered in these studies.

perceived irrelevance

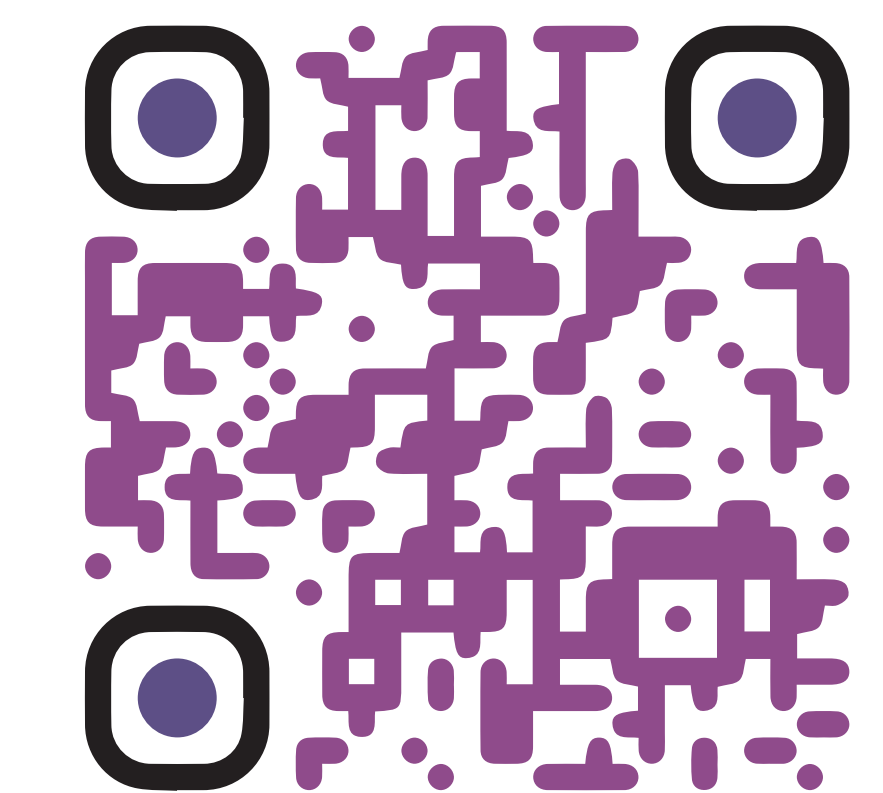
inadequate ecosystem

unfamiliar concepts



complex theory

"weird" design



Check out my profile for the full story!