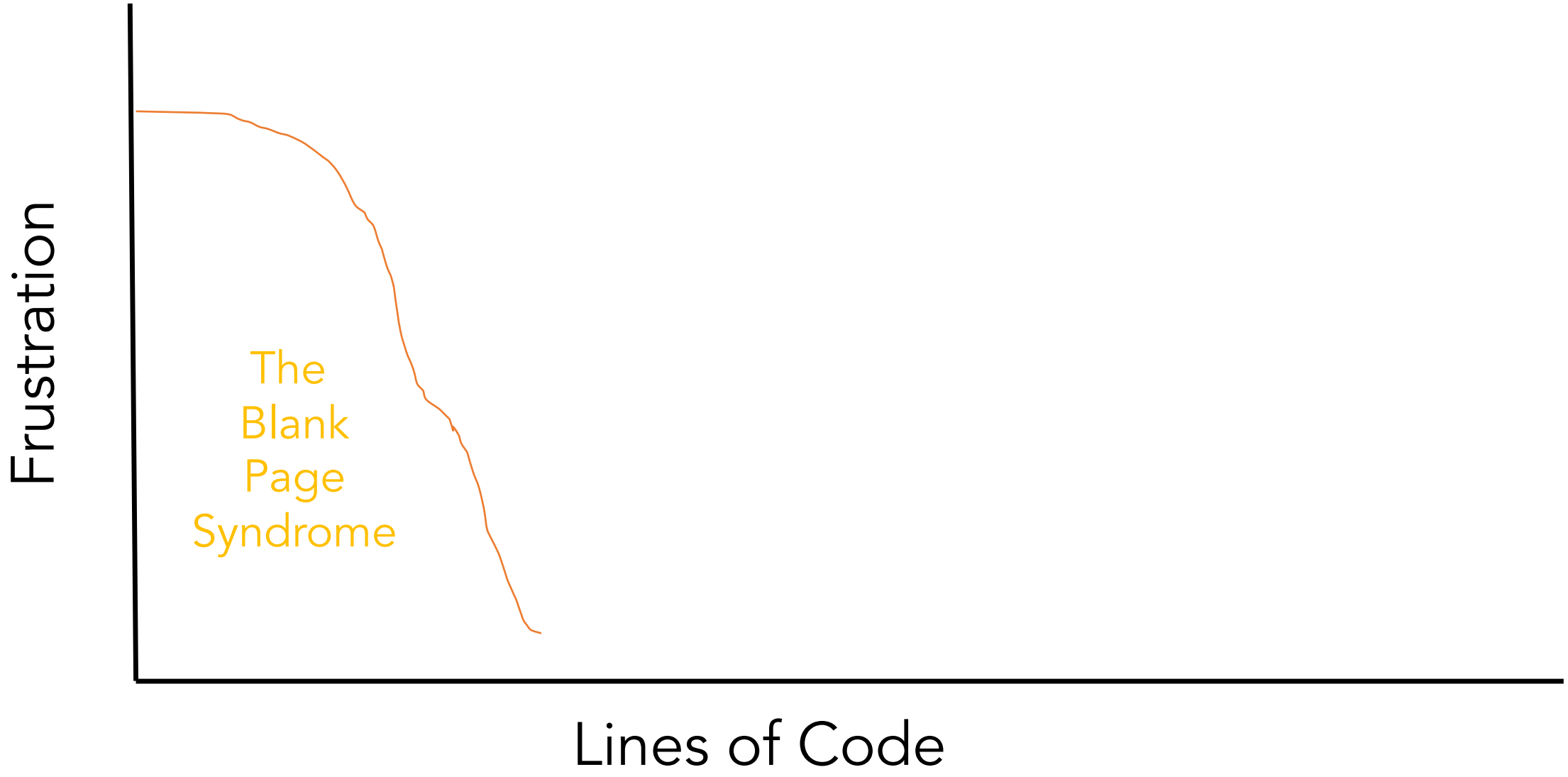
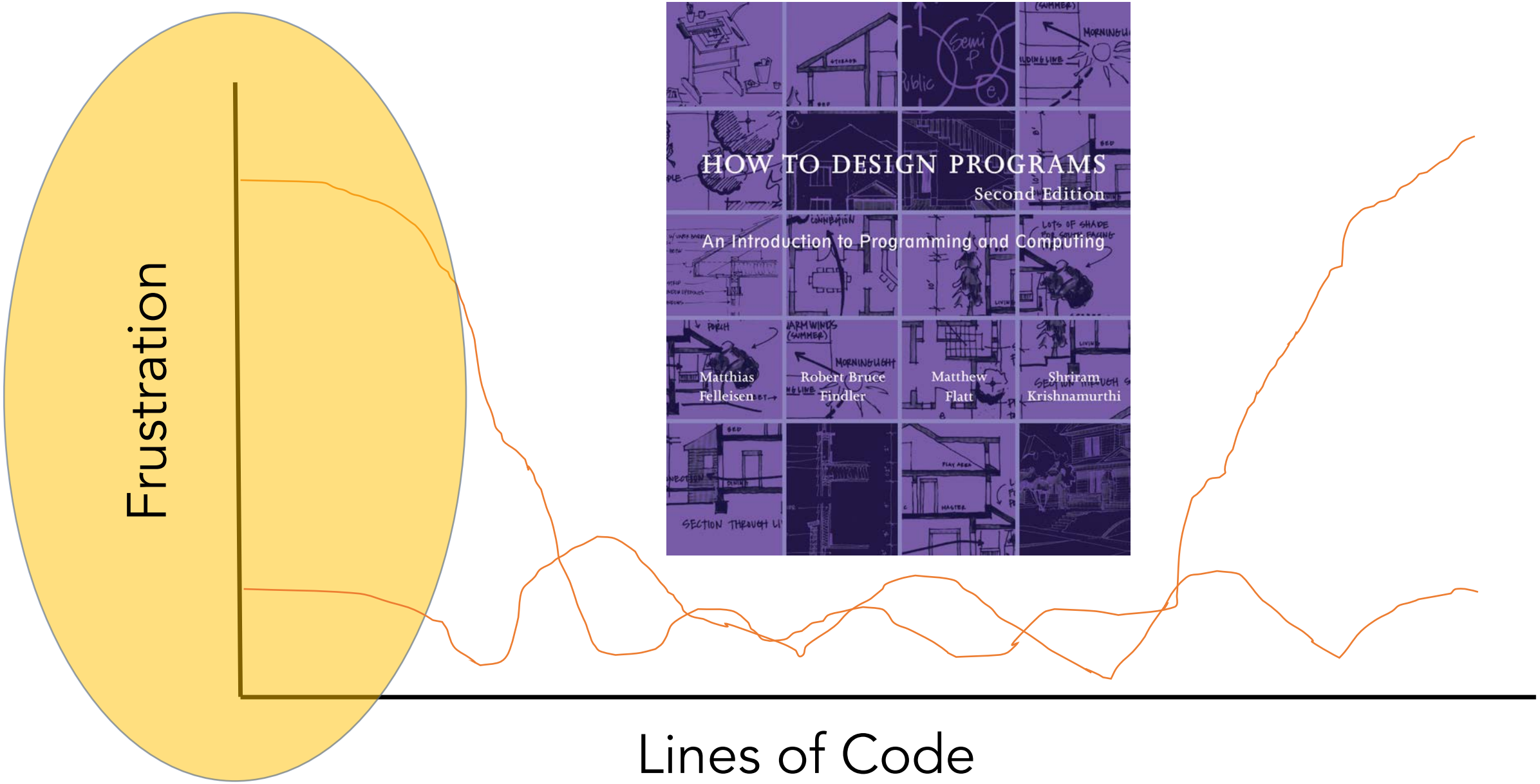
How to Plan Programs

Shriram Krishnamurthi, *Brown University*

Joint work primarily with

Kathi Fisler
Siddhartha Prasad
Elijah Rivera
Jack Wrenn


Frustration
The
Blank
Page
Syndrome
Lines of Code


Frustration
HOW TO DESIGN PROGRAMS
Second Edition
An Introduction to Programming and Computing
Matthias Felleisen
Robert Bruce Findler
Matthew Flatt
Shriram Krishnamurthi
Lines of Code

Understanding Problems Before Programming

Exercise

Implement a median function

Given a list of numbers, compute its median

Question: What could go wrong?

- Not accounting for empty input
- Not accounting for even-length lists (left median? right median? average?)

- Not accounting for unsorted input

- Confusing median for mean
- Confusing median for mode
- …

Students often solve the wrong problem

Autograding:

- Comes too late
- Fear of change (sunk costs)
- Learning objectives lost

We keep focusing on understanding pro**gra**ms
We don't focus enough on understanding pro**ble**ms

Standard solution:

Express the problem in your own words.

Problem:

Students don't have their "own" words!

Exercise:

Provide examples (input/output pairs)

for a `median` function

```
median([list: 1, 2, 3]) is 2
median([list: 1, 2, 3, 4, 5]) is 3

median([list: 1, 2, 3, 4]) is …

median([list: 1, 3, 2]) is …
```

Examples provide a *concrete* form of "in your own words"
Examples are also useful as *test cases*

But *inert* examples are useless (and students won't write them [Castro & Fisler])

Idea

Run examples against known correct & known buggy implementations
(We can do this *before they've written any code*)

"Semantic mutation testing":
The buggy implementations correspond to known *misconceptions*

Test suites are classifiers

```
median([list: 1, 2, 3]) is 2
median([list: 1, 2, 3, 4, 5]) is 3

median([list: 1, 2, 3, 4]) is 2.5

median([list: 1, 3, 2]) is 3
```

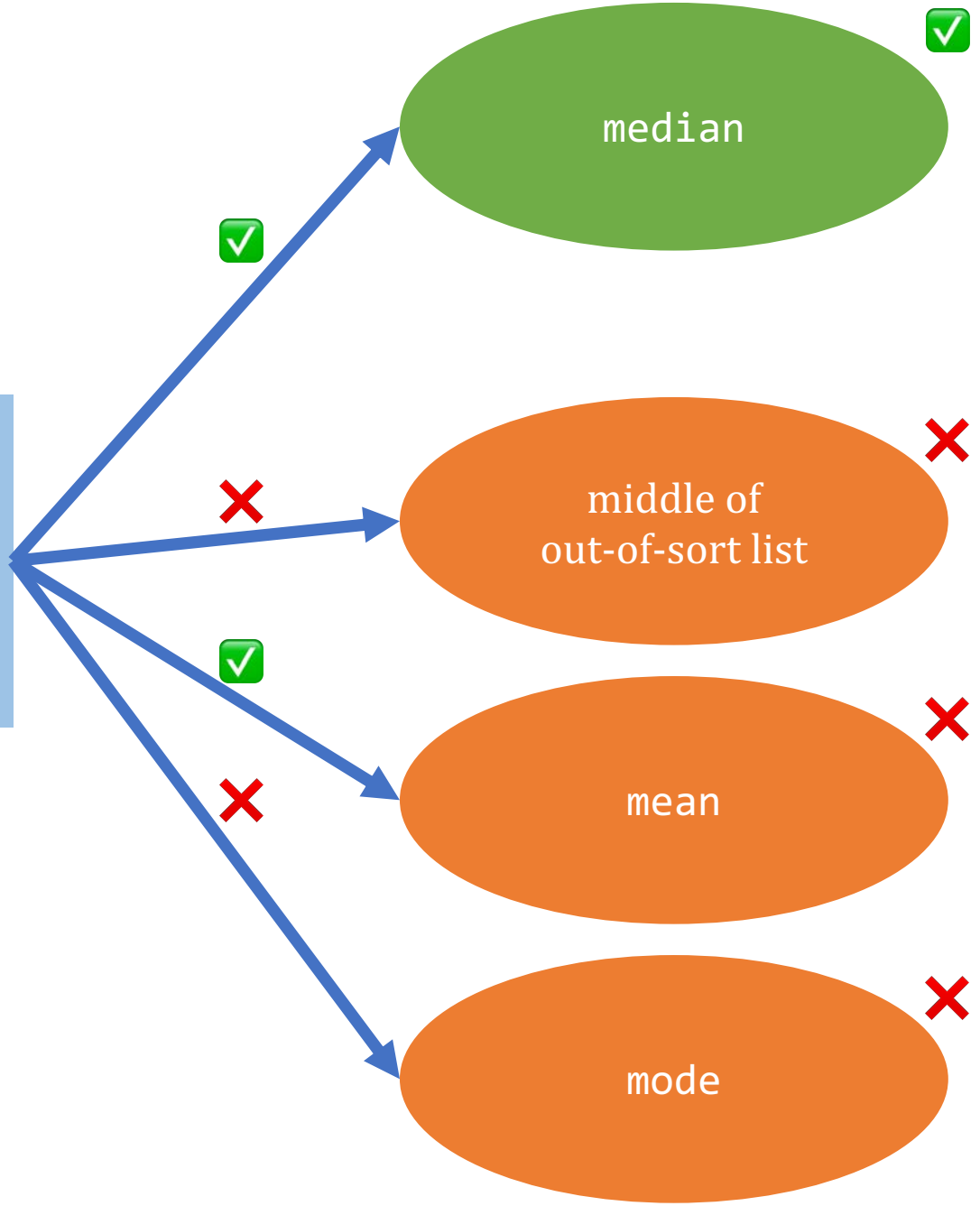
This is “autograding” — but before any code has been written

Test suites are classifiers

```
median([list: 1, 2, 3]) is 2
median([list: 1, 2, 3, 4, 5]) is 3

median([list: 1, 2, 3, 4]) is 2.5

median([list: 1, 3, 2]) is 2
```



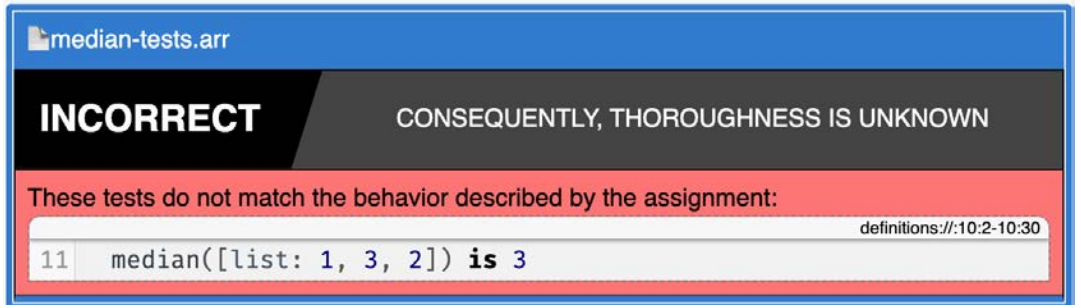
validity

thoroughness

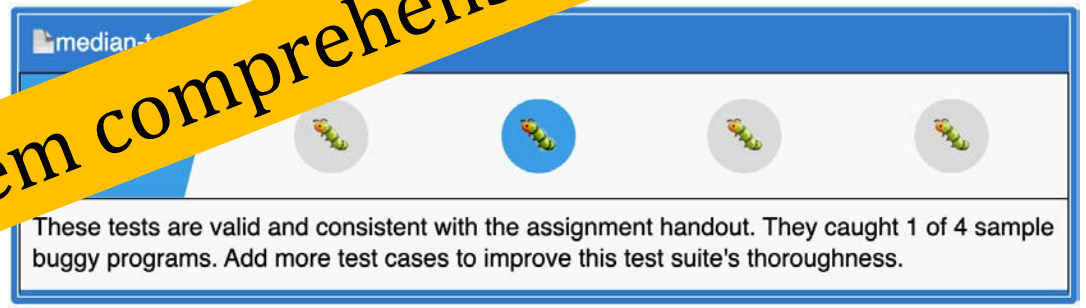
Ideal test suite:
Passes correct impl(s)
Catches all buggy impls

Pyret

```
median([list: 1, 3, 2]) is 3
```

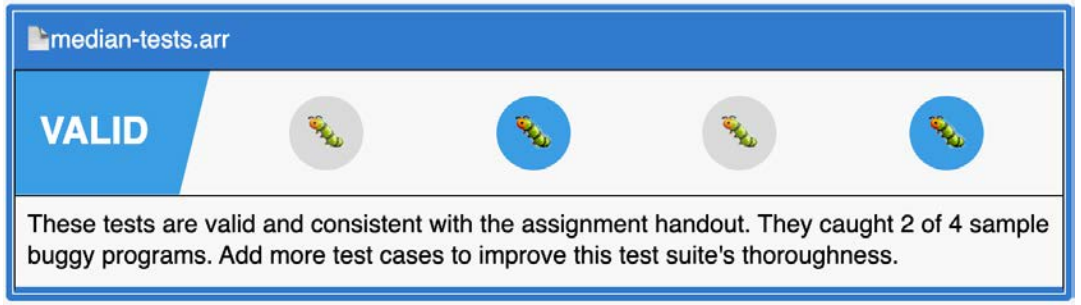



```
median([list: 1, 2, 3]) is 2
```



"24 hour TA" (for problem comprehension)

```
median([list: 1, 2, 3, 7, 10]) is 3
```



“Semantic mutation testing”:

The buggy implementations
correspond to
known *misconceptions*

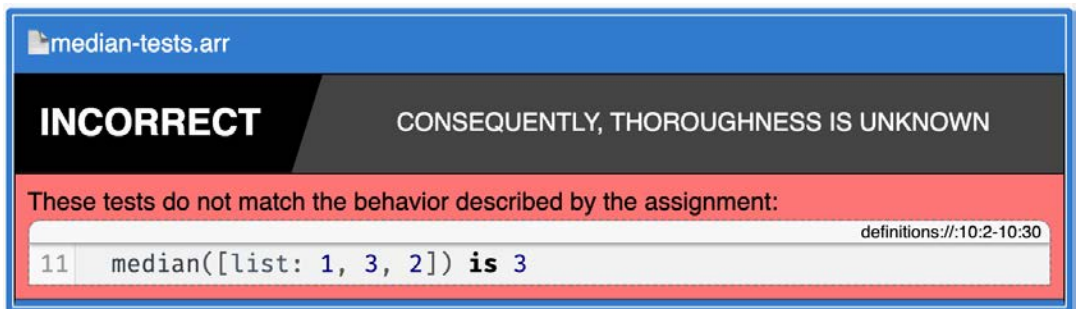
How do we know this?

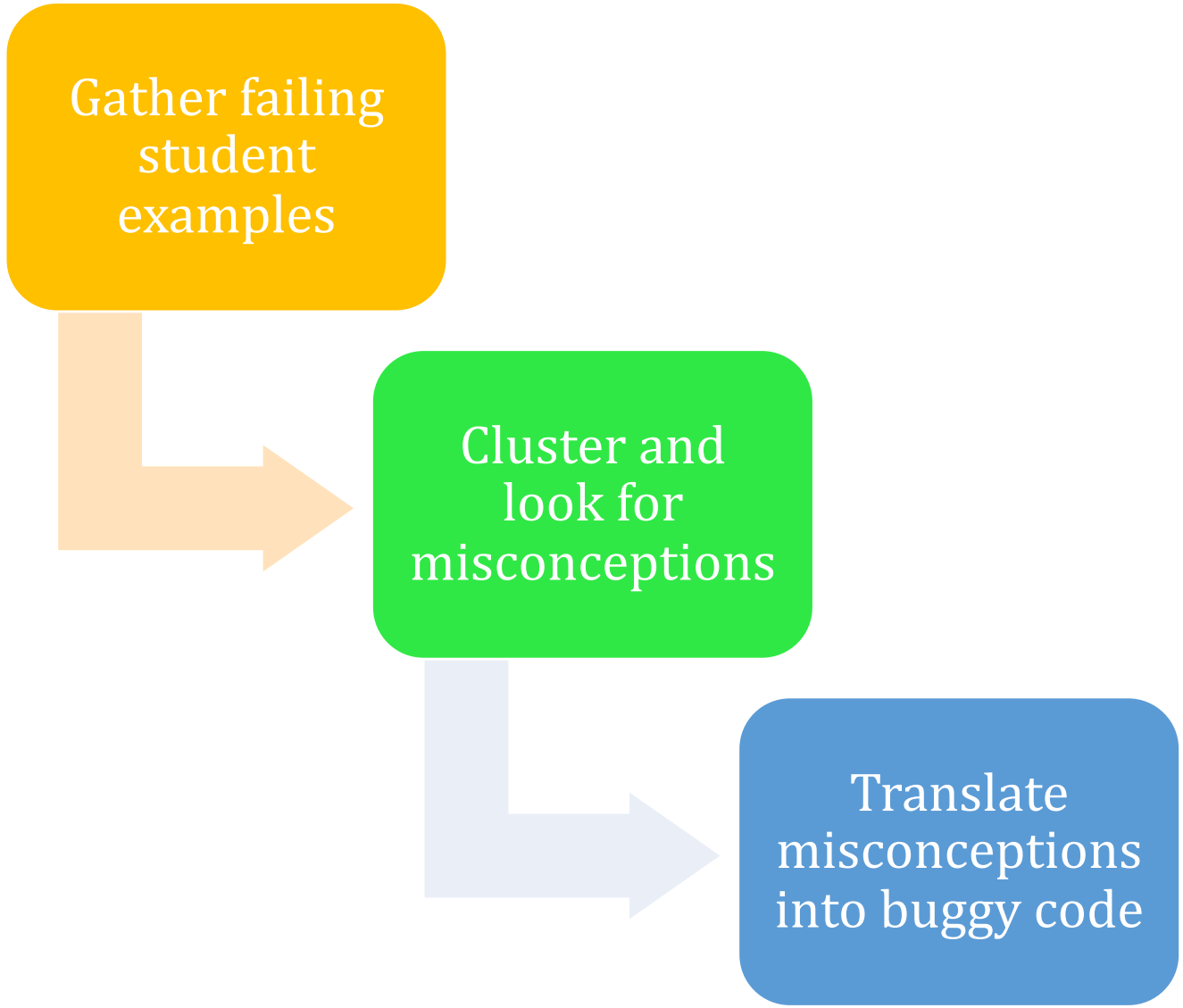
Experts curate the mutants
based on experience, student questions, etc.

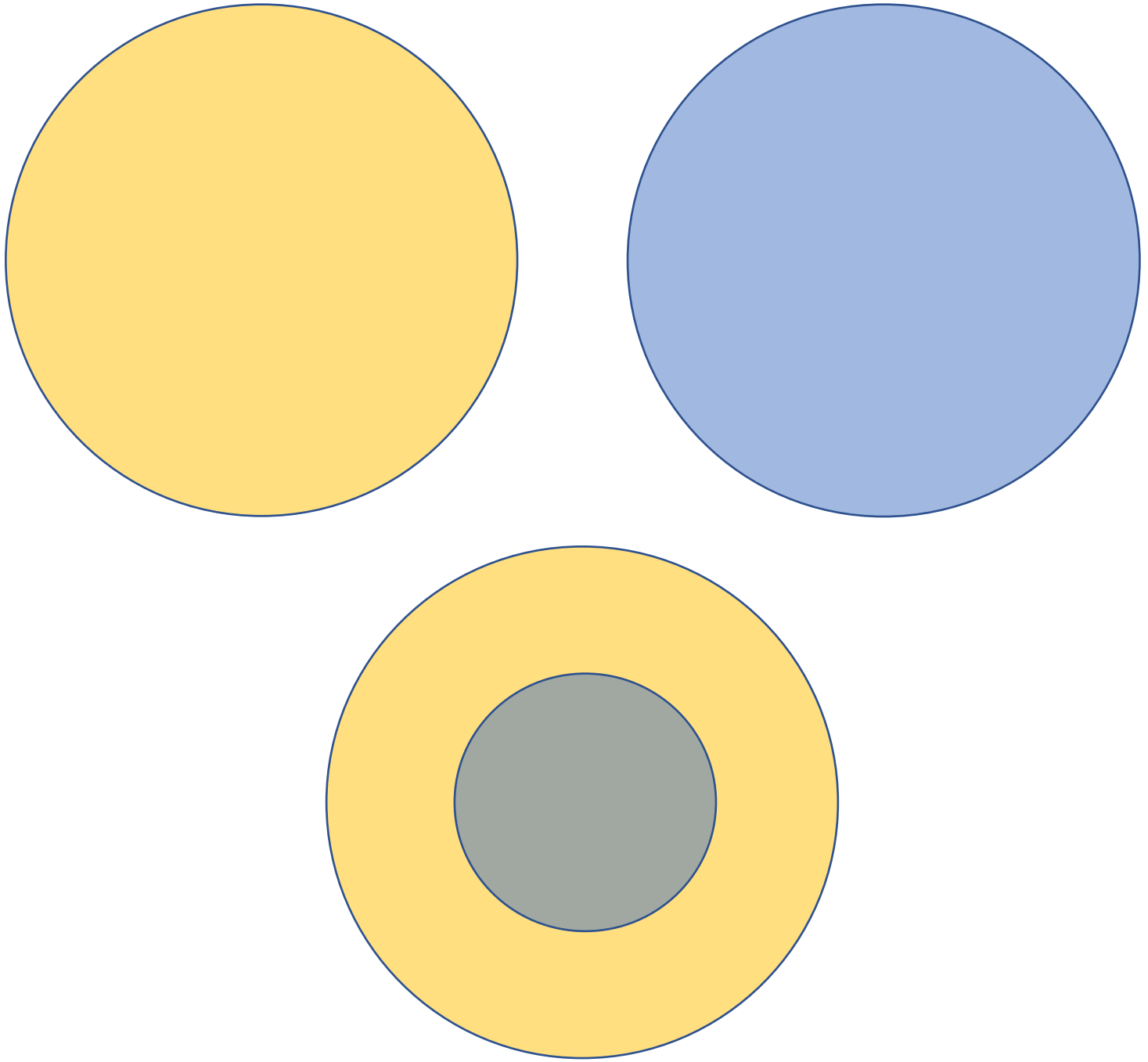
But the *expert blind spot* is a real concern!

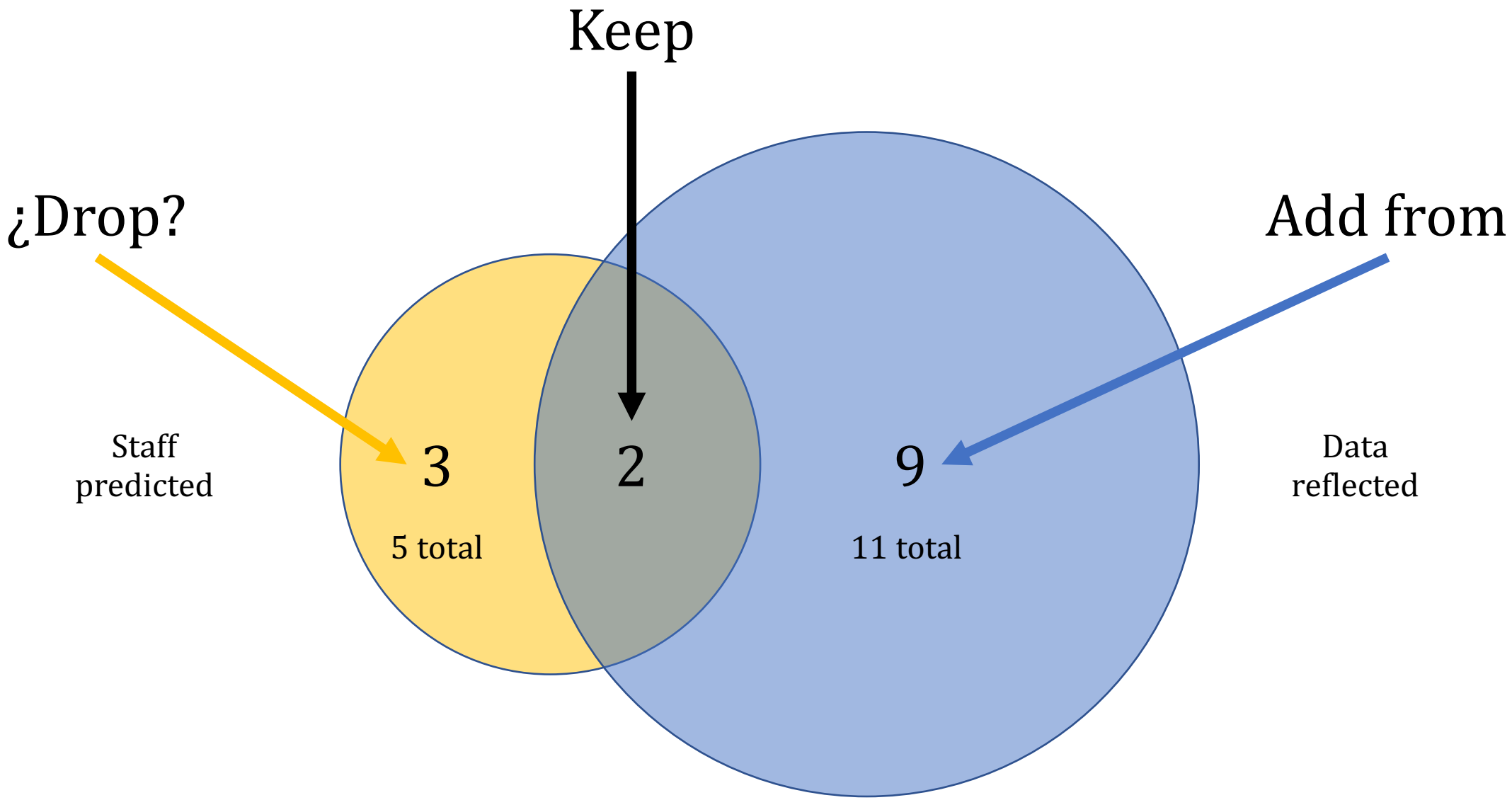
Pyret

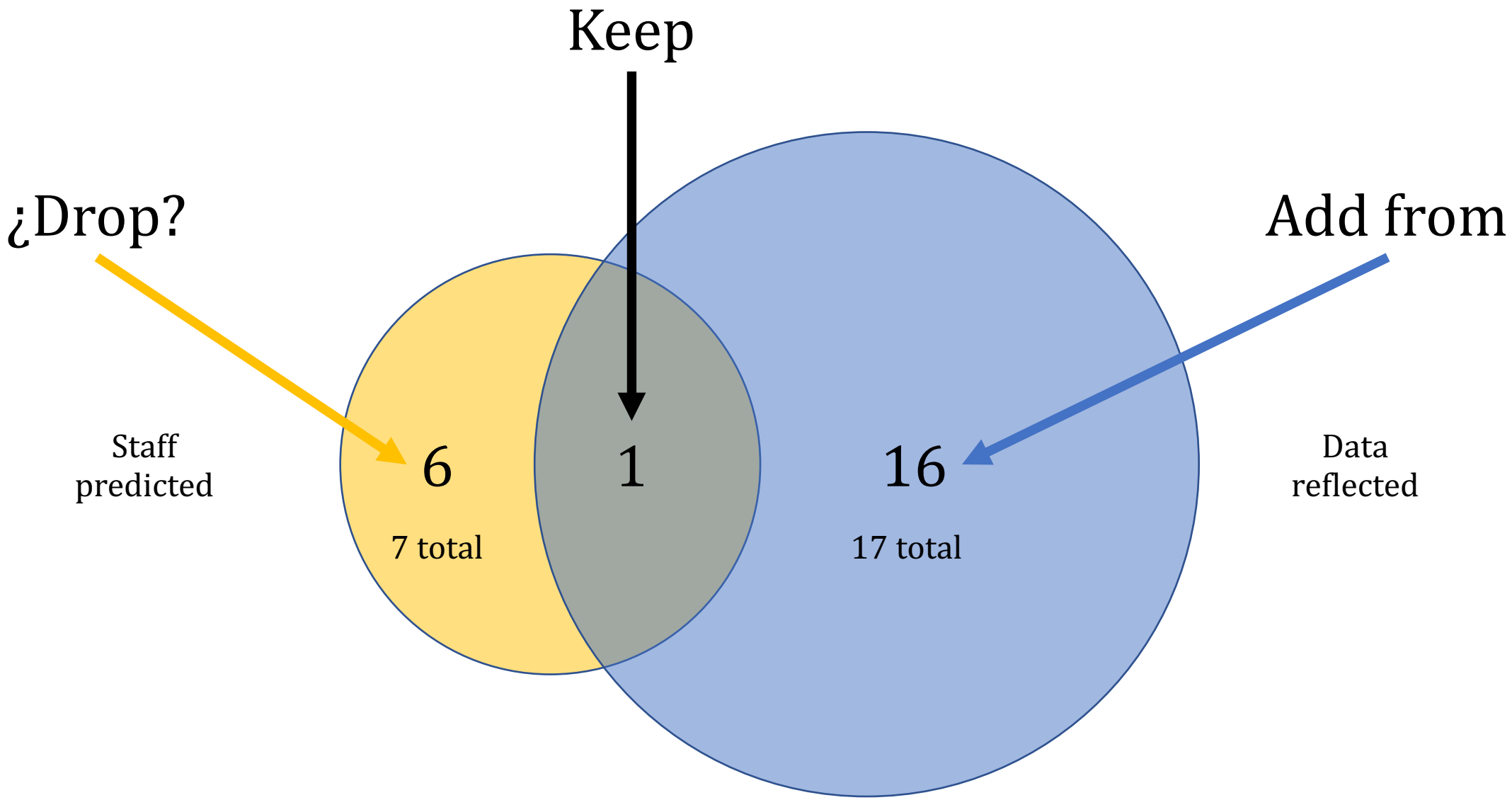
```
median([list: 1, 3, 2]) is 3
```




Gather failing student examples
Cluster and look for misconceptions
Translate misconceptions into buggy code


Staff predicted
Data reflected


Keep
¿Drop?
Add from
Staff predicted
3
5 total
2
9
11 total
Data reflected


Keep
¿Drop?
Add from
Staff predicted
6
7 total
1
16
17 total
Data reflected

Classsourcing

Identifies misconceptions from failing examples

Works around expert blind-spots

Relatively lightweight

Summary

Students program before understanding the problem

We need lightweight ways to help them debug understanding

Examples are very good for this

Examples can be operationalized via “semantic mutation testing”

Student mistakes can be mined for insight

Executable Examples for Programming Problem Comprehension

John Wrenn
Computer Science Department
Brown University
Providence, Rhode Island, USA
jswrenn@cs.brown.edu

Shriram Krishnamurthi
Computer Science Department
Brown University
Providence, Rhode Island, USA
sk@cs.brown.edu

Will Students Write Tests Early Without Coercion?*

John Wrenn
Computer Science Department
Brown University
Providence, Rhode Island, USA
jswrenn@cs.brown.edu

Shriram Krishnamurthi
Computer Science Department
Brown University
Providence, Rhode Island, USA
sk@cs.brown.edu

Making Hay from Wheats: A Classsourcing Method to Identify Misconceptions

Siddhartha Prasad
Computer Science Department
Brown University
Providence, Rhode Island, USA
siddhartha_prasad@brown.edu

Ben Greenman
Computer Science Department
Brown University
Providence, Rhode Island, USA
benjamin.l.greenman@gmail.com

Tim Nelson
Computer Science Department
Brown University
Providence, Rhode Island, USA
timothy_nelson@brown.edu

John Wrenn
Computer Science Department
Brown University
Providence, Rhode Island, USA
jack@wrenn.fyi

Shriram Krishnamurthi
Computer Science Department
Brown University
Providence, Rhode Island, USA
shriram@brown.edu

Who Tests the Testers?*

Avoiding the Perils of Automated Testing

John Wrenn
Computer Science
Brown University
USA
jswrenn@cs.brown.edu

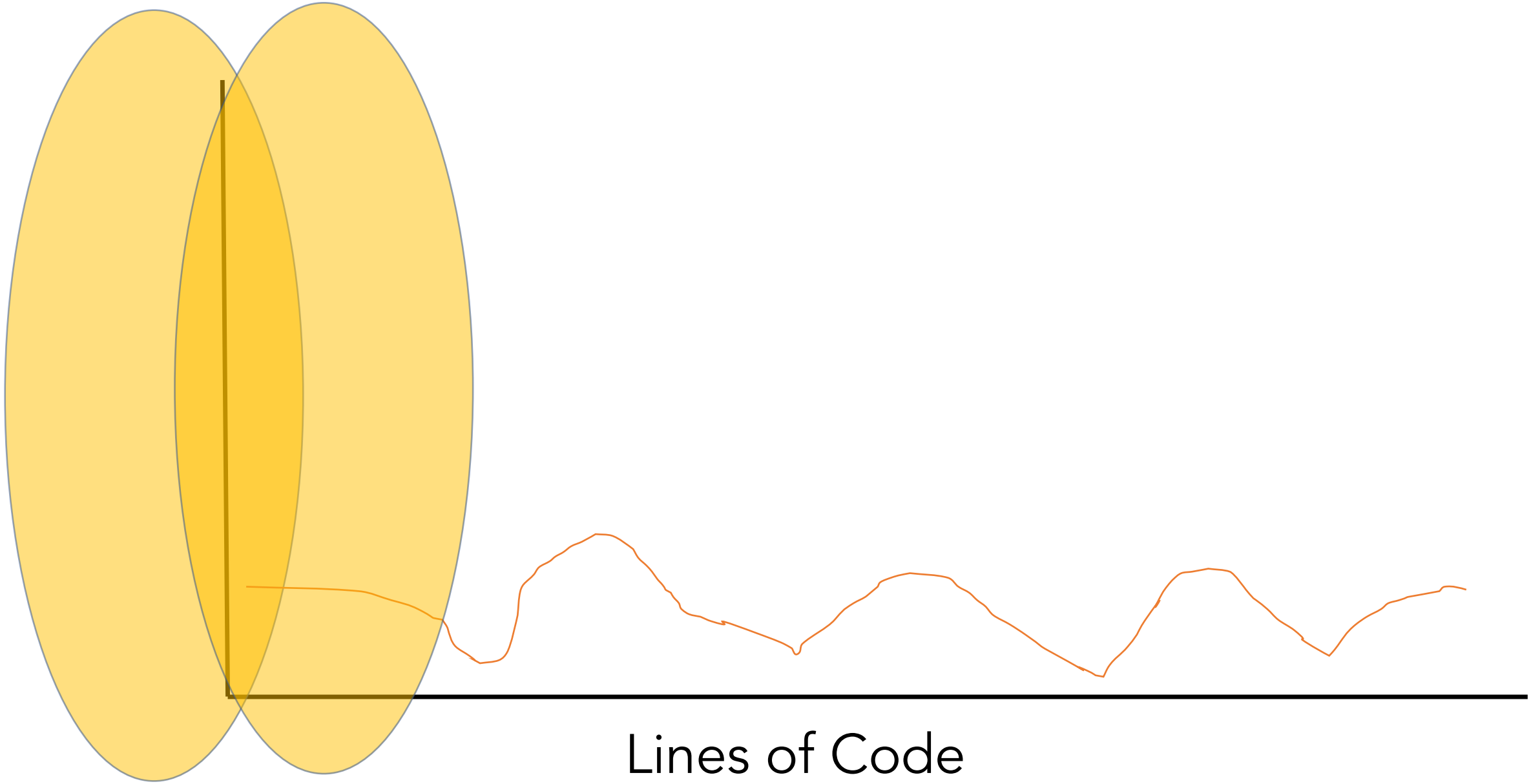
Shriram Krishnamurthi
Computer Science
Brown University
USA
sk@cs.brown.edu

Kathi Fisler
Computer Science
Brown University
USA
kfisler@cs.brown.edu

Reading Between the Lines: Student Help-Seeking for (Un)Specified Behaviors

JOHN WRENN, Brown University, USA
SHRIRAM KRISHNAMURTHI, Brown University, USA

Pyret


Lines of Code

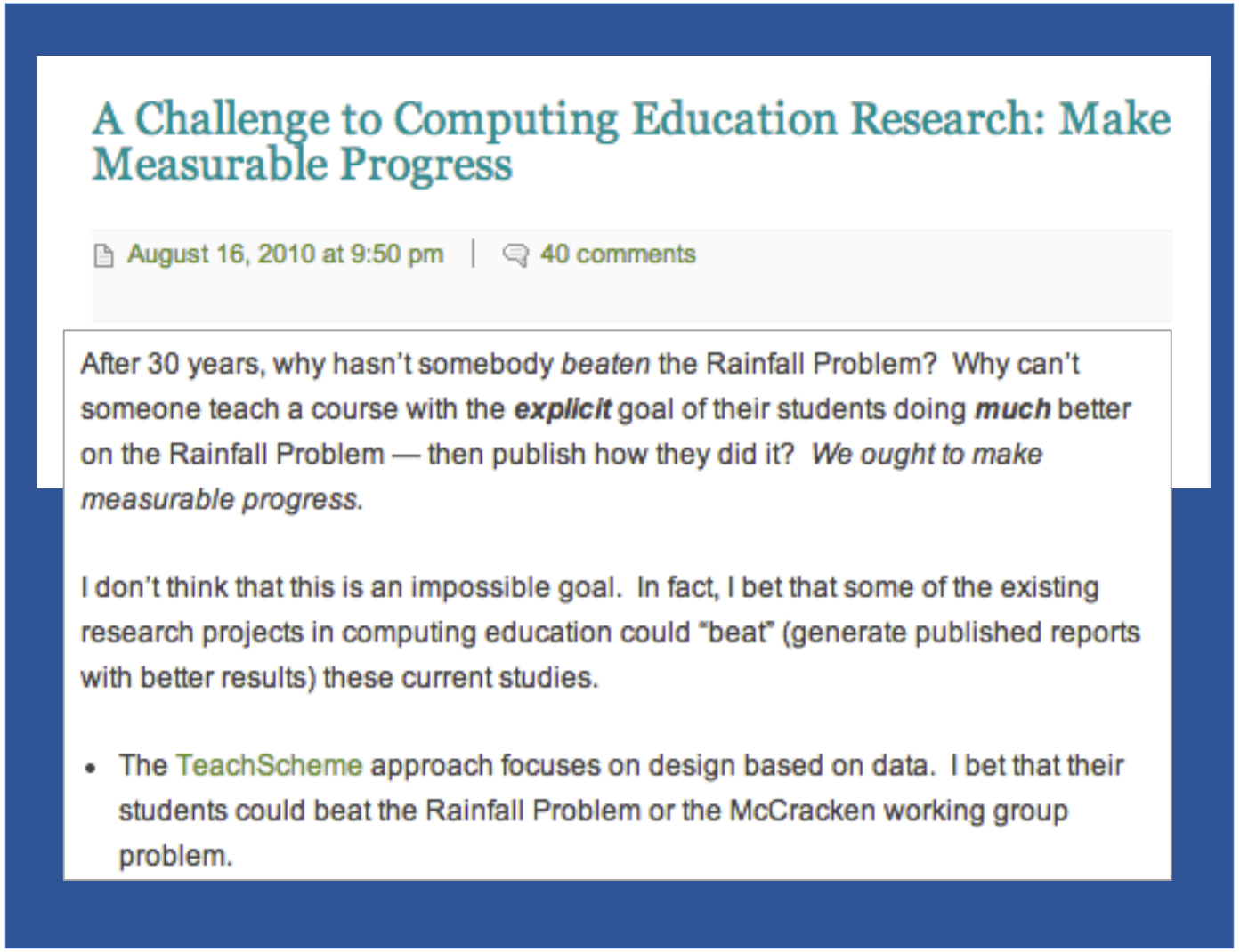
Planning Programs Before Programming

Write a program that takes a list/array of numbers and produces the average of the non-negative numbers that occur before (an optional) -999

Ignore I/O – just take the list/array as input

Use any language you wish

On Mark Guzdial's Blog



A Challenge to Computing Education Research: Make Measurable Progress

August 16, 2010 at 9:50 pm | 40 comments

After 30 years, why hasn't somebody *beaten* the Rainfall Problem? Why can't someone teach a course with the ***explicit*** goal of their students doing ***much*** better on the Rainfall Problem — then publish how they did it? *We ought to make measurable progress.*

I don't think that this is an impossible goal. In fact, I bet that some of the existing research projects in computing education could "beat" (generate published reports with better results) these current studies.

- The TeachScheme approach focuses on design based on data. I bet that their students could beat the Rainfall Problem or the McCracken working group problem.

Write a program that takes a list/array of numbers and produces the average of the non-negative numbers that occur before (an optional) -999

- truncate at sentinel
- ignore negatives
- sum the data
- count the data
- compute average
- handle empty data (division by zero)

[Fisler 2014] studied students at 4 unis/5 courses using HtDP

Findings:

Students with clear structure had far fewer errors

Students made good use of built-in and higher-order functions

Students showed a much greater diversity of structures
than traditional in imperative programming

Guzdial: this “beat” rainfall

Rainfall: Write a program that takes a list/array of rainfall values (numbers) and produces the average of the non-negative numbers that occur before (an optional) -999

Palindrome: standard definition, but ignore whitespace, punctuation, and capitalization

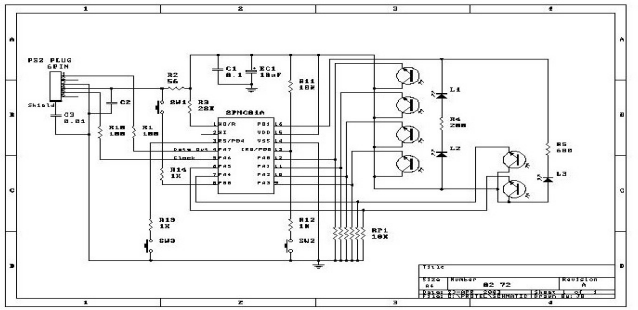
Adding Machine: given list of numbers, return list of sums of each sublist separated by zeros; stop after two consecutive zeros

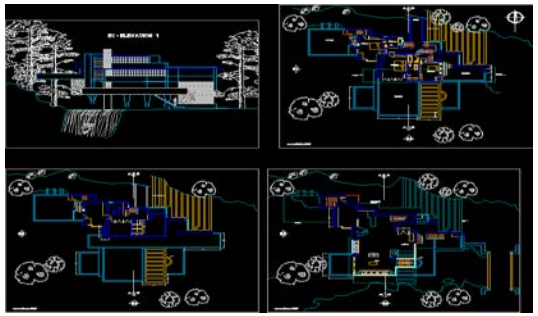
e.g.: `[3,0,2,5,0,0,9]` → `[3,7]`

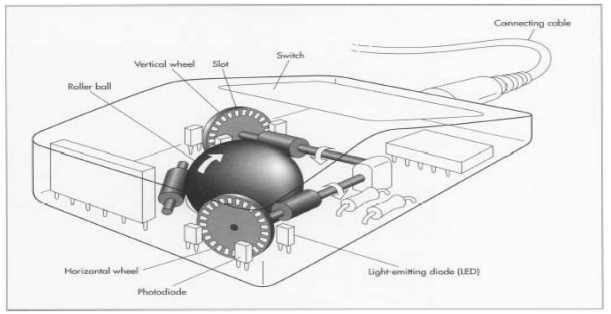
Shopping Cart: compute total cost of a cart (list of items), giving one discount based on the number of hats in the cart and another based on the total price of shoes in the cart

All the prior research has been in the “backward” direction:
Have students write programs, try to determine plans

What if we go in the “forward” direction?
Have students write plans *before* programming?






Connecting cable
Vertical wheel
Slot
Switch
Roller ball
Horizontal wheel
Photodiode
Light-emitting diode (LED)

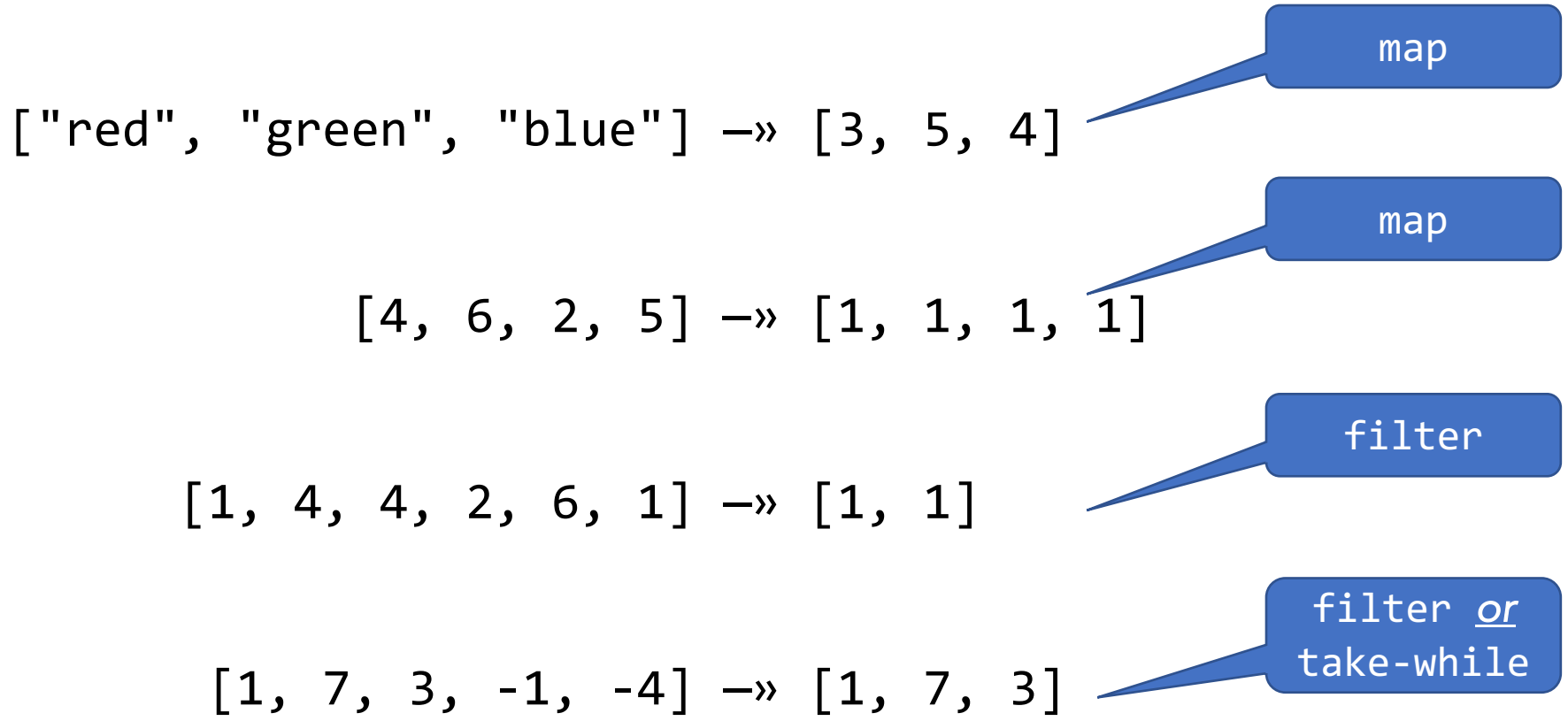
Two Different Views of HOFs

Common in textbooks (e.g., *HtDP*)

HOFs as abstractions of *code*
(i.e., uniformity over several traversal/processing patterns)

View we're trying to develop

HOFs as abstractions of ***behavior***
(i.e., uniformity over several data transformations)


["red", "green", "blue"] –» [3, 5, 4]
map
[4, 6, 2, 5] –» [1, 1, 1, 1]
map
[1, 4, 4, 2, 6, 1] –» [1, 1]
filter
[1, 7, 3, -1, -4] –» [1, 7, 3]
filter or take-while

	Output Type	Output Element Type	Output Length
`map`	list	can differ	same
`filter`	list	same	<=
`ormap`	bool		

Important from both **plan composition** and **data science** perspectives

Functional programming (FP) provides a rich set of tools for reducing duplication in your code. The goal of FP is to make it easy to express repeated actions using high-level verbs. I think that learning a little about FP is really important for data scientists, because it's a really good fit for many problems that you'll encounter in practice.

—Hadley Wickham, *Advanced R*, etc.

Three Kinds of Activities

Clustering:
given i/o pairs, put them in equivalent groups

Labeling:
given *pictorial* versions, indicate intended functions

Classification:
for each i/o pair, indicate which function(s) can do this

Takeaways

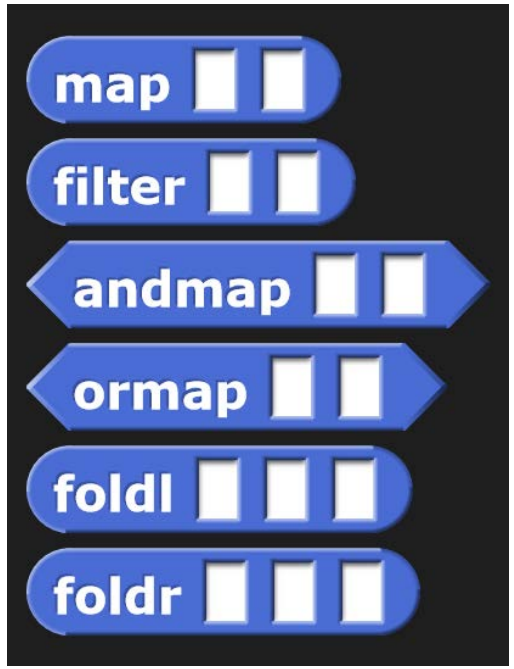
Students did increasingly well at our tasks;
we also found some common errors

Students do confuse distinct functions with similar features

Clustering and *classification* are very useful activities

We've created useful instruments
for both research and teaching

Onward to planning…




```
+valid-words+ words + letters +
report
filter lambda function that checks if the string is only made up of the characters given strings
```

```
+valid-words+ words + letters +
report filter andmap is a char a valid letter? letters of word words
```

```
+valid-words+ words + letters +
report
filter
andmap (Helper function: is-valid-char? Use "member" to determine if a char is in LETTERS)
words
(Each word, deconstructed into a list using string->list)
```

Modernizing Plan-Composition Studies

Kathi Fisler
WPI
kfisler@cs.wpi.edu

Shriram Krishnamurthi
Brown University
sk@cs.brown.edu

Janet Siegmund
University of Passau
siegmunj@
fim.uni-passau.de

Developing Behavioral Concepts of Higher-Order nctions

Shriram Krishnamurthi
Brown University
Providence, RI, USA

, USA

We're looking for collaborators!

Plan Compo gher-Order Functions

Elijah Ri
eeriver
Brown
Providen , USA

hriram Krishnamurthi
shriram@brown.edu
Brown University
Providence, RI, USA

Robert Goldstone
rgoldsto@indiana.edu
Indiana University
Bloomington, IN, USA

Structural versus Pipeline Composition of Higher-Order Functions (Experience Report)

ELIJAH RIVERA, Brown University, USA
SHRIRAM KRISHNAMURTHI, Brown University, USA

New Curricular Horizons

Data Structure Progress Since the 1970s

1970s	Pascal	arrays
1980s	C	arrays
1990s	C++	arrays
2000-2015	Java	ArrayList
2015-now	Python	associative arrays

Whereas in Functional Programming…

1960s	Lisp	lists
1980s	Scheme	lists
1990s	ML	lists
2000s	Haskell	lists
Now?	Lean/Coq/…	Building artisanal lists inductively

GPT-3

LowCode/NoCode

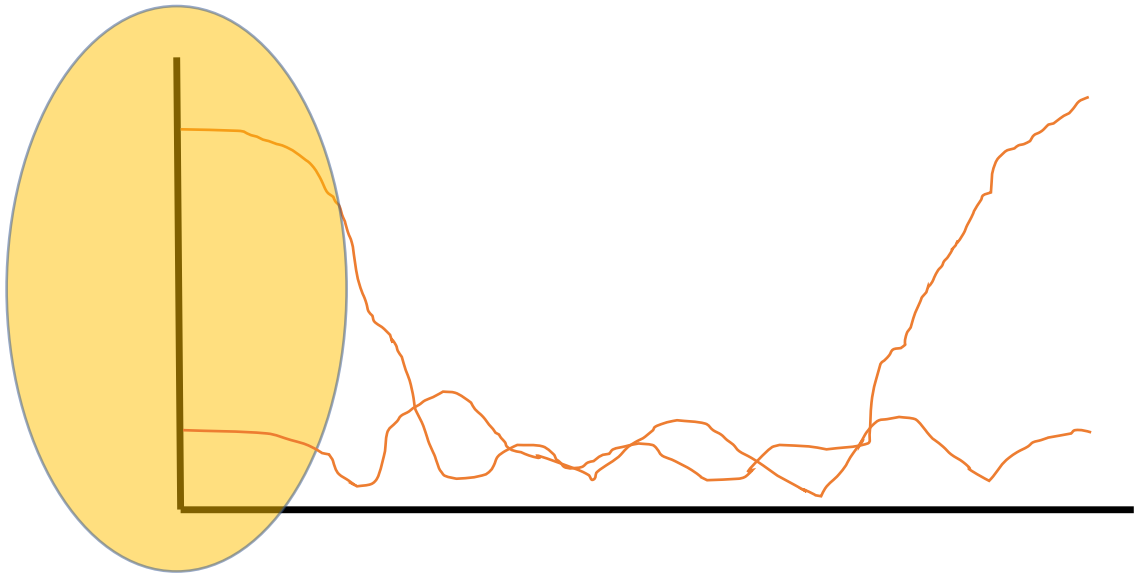
Data-centric algorithms

StackOverflow

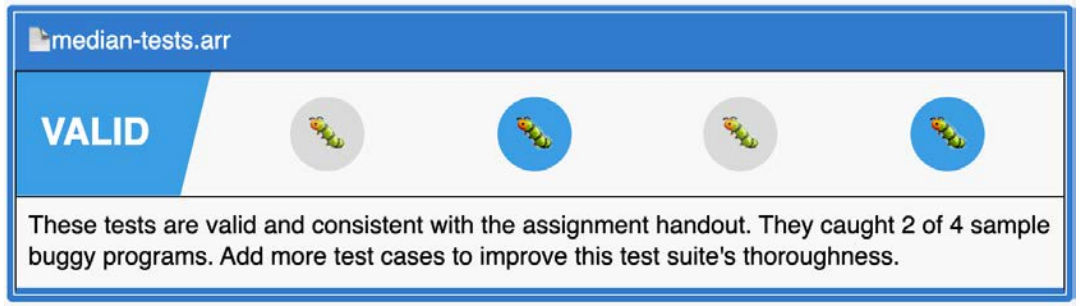
Program synthesis

Property-based testing

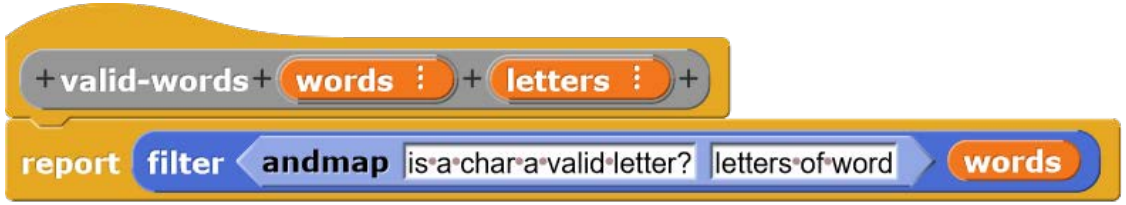
Verification in programming



Understanding Problems Before Programming



Planning Problems Before Programming



shriram@gmail.com

@shriramk@mastodon.social