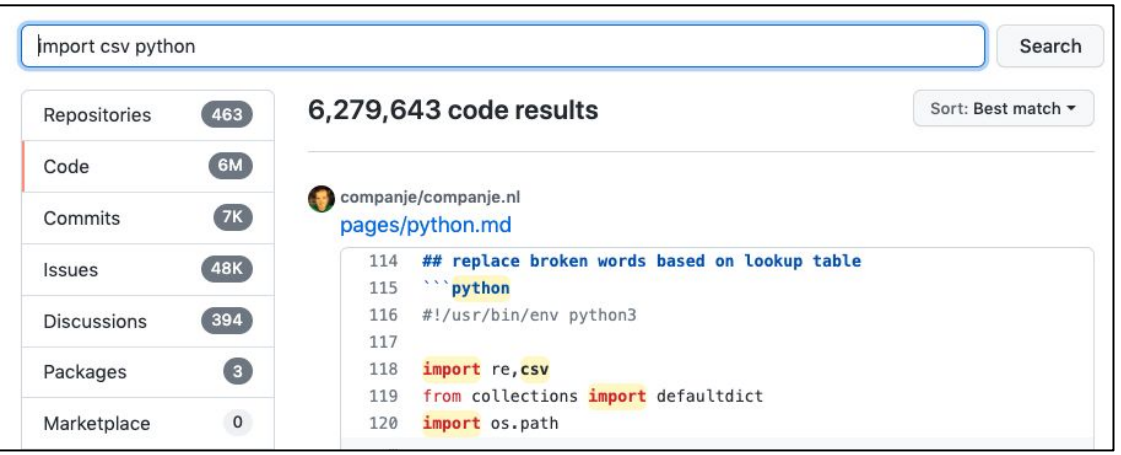
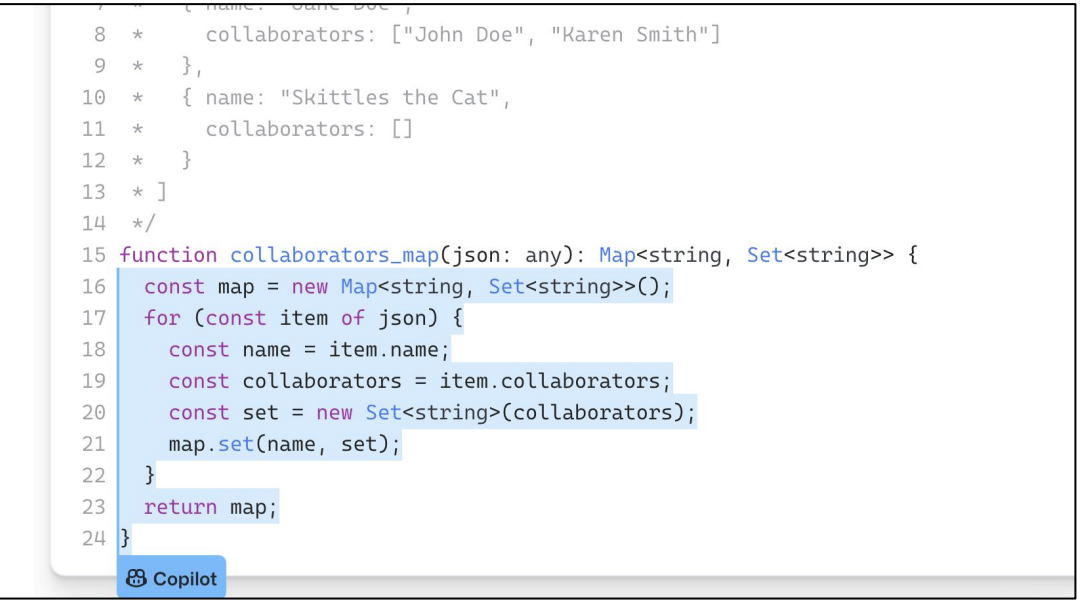
Code Search and Comprehension in Software Engineering

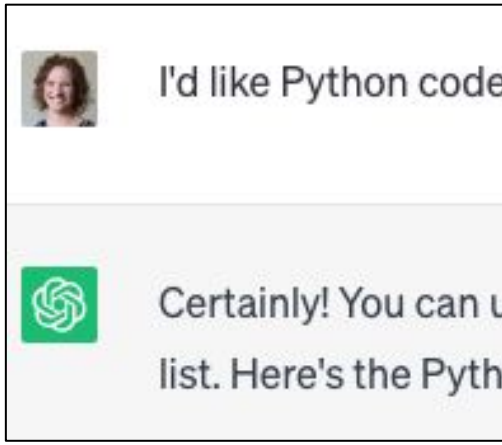
Dr. Kathryn (Katie) Stolee
Associate Professor *with tenure*
North Carolina State University

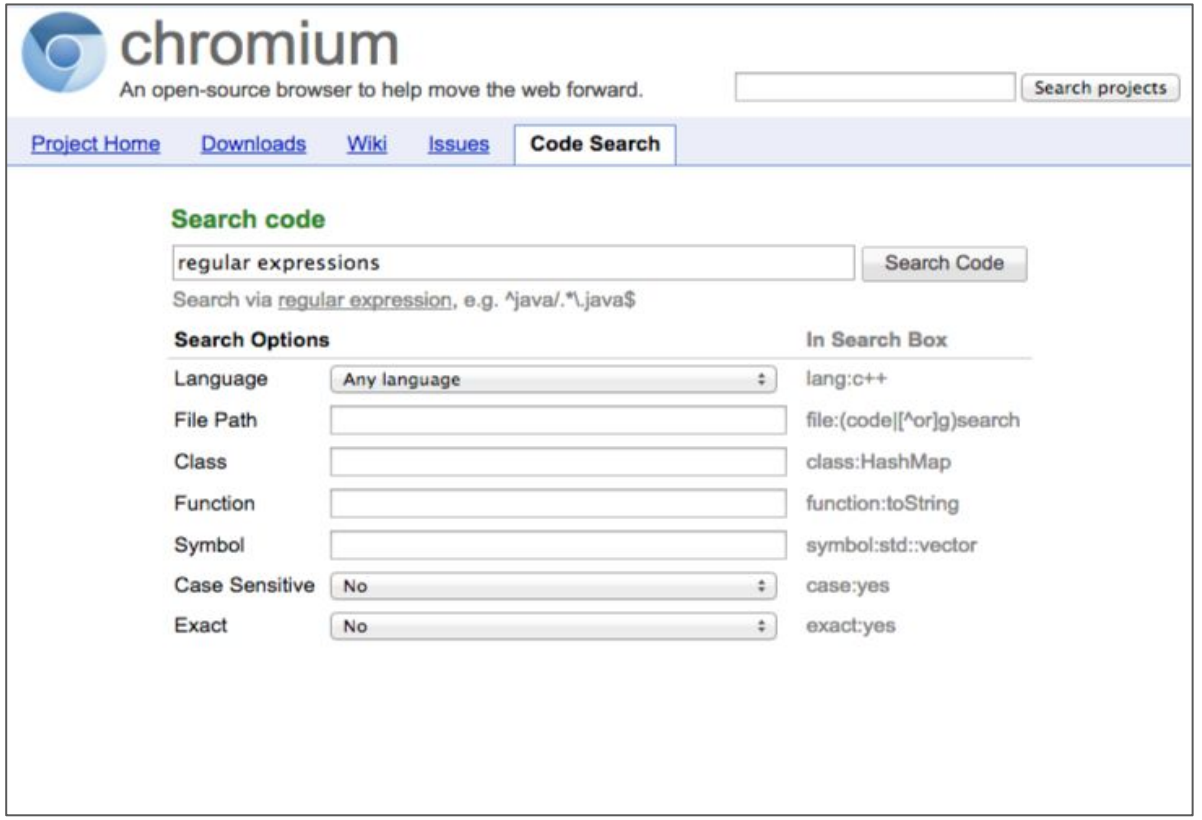
NC STATE UNIVERSITY


import csv python
Search
Repositories 463
Code 6M
Commits 7K
Issues 48K
Discussions 394
Packages 3
Marketplace 0
6,279,643 code results
Sort: Best match
companje/companje.nl
pages/python.md
114 ## replace broken words based on lookup table
115 ```python
116 #!/usr/bin/env python3
117
118 import re,csv
119 from collections import defaultdict
120 import os.path


8 * collaborators: ["John Doe", "Karen Smith"]
9 * },
10 * { name: "Skittles the Cat",
11 * collaborators: []
12 * }
13 *]
14 */
15 function collaborators_map(json: any): Map<string, Set<string>> {
16 const map = new Map<string, Set<string>>();
17 for (const item of json) {
18 const name = item.name;
19 const collaborators = item.collaborators;
20 const set = new Set<string>(collaborators);
21 map.set(name, set);
22 }
23 return map;
24 }
Copilot


Google
Google Search
I'm Feeling Lucky


I'd like Python code
Certainly! You can u
list. Here's the Pyth


chromium
An open-source browser to help move the web forward.
Search projects
Project Home
Downloads
Wiki
Issues
Code Search
Search code
regular expressions
Search Code
Search via regular expression, e.g. ^java/.*\.java$
Search Options
In Search Box
Language
Any language
lang:c++
File Path
file:(code|[^or]g)search
Class
class:HashMap
Function
function:toString
Symbol
symbol:std::vector
Case Sensitive
No
case:yes
Exact
No
exact:yes

Code search is frequent

- ~12x per developer per day [in 2012]
- Search sessions involve multiple queries
- Code search with Google takes more time, more clicks, and more query reformulation than non-code search

Two Most Common Needs

1. Example Code, **how** to do something (33%)

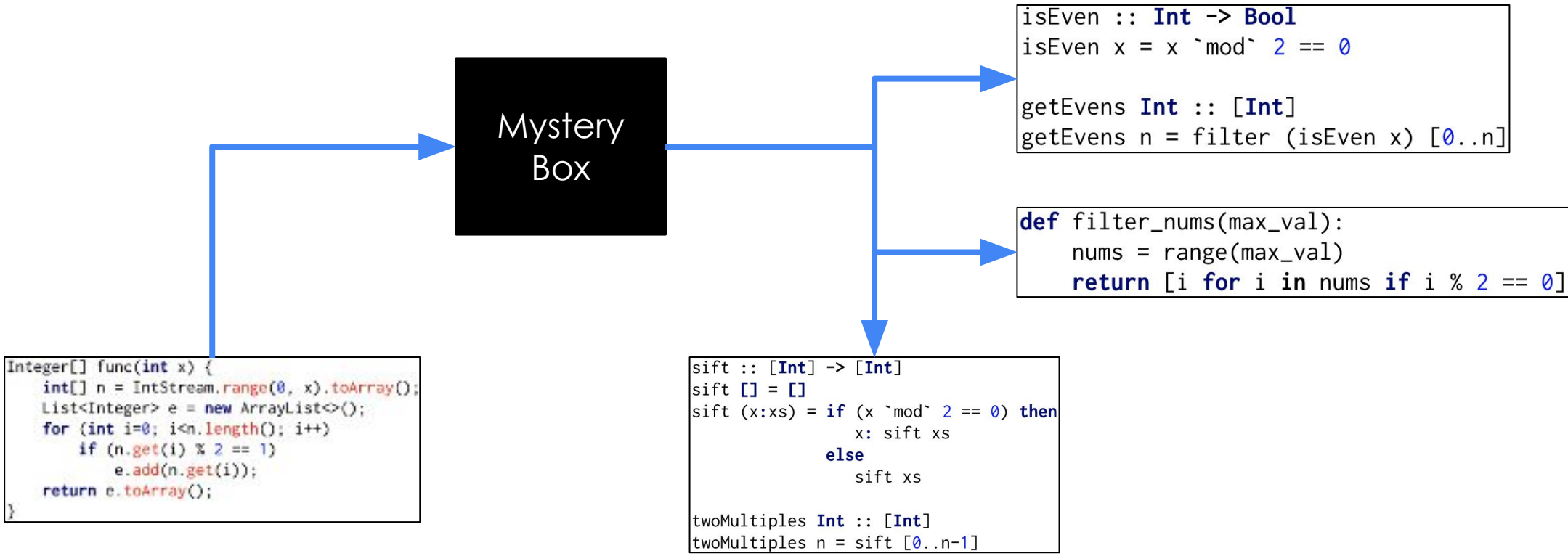
2. Explaining **what** it does (26%)

FIND IT

THINK IT

CHOOSE IT

Code-to-Code Search



The Halting Problem 😱

IT MAY NEVER WORK IN THEORY.

Code-to-code Search

```java
List<Integer> getOdds(int max) {
    List<Integer> odds = new ArrayList<>();
    for(int i = 0; i < max; i++)
        if (i % 2 == 1)
            odds.add(i);
    return odds;
}
```

Java: **for** loop to populate array of ***odd*** numbers

```haskell
sift :: [Int] -> [Int]
sift [] = []
sift (x:xs) = if (x `mod` 2 == 0) then
                 x: sift xs
              else
                 sift xs

twoMultiples Int :: [Int]
twoMultiples n = sift [0..n-1]
```

Haskell: List of ***even*** numbers using recursion

```haskell
isEven :: Int -> Bool
isEven x = x `mod` 2 == 0

getEvens Int :: [Int]
getEvens n = filter (isEven x) [0..n]
```

Haskell: List of ***even*** numbers using chaining

```java
Integer[] func(int x) {
    int[] n = IntStream.range(0, x).toArray();
    List<Integer> e = new ArrayList<>();
    for (int i=0; i<n.length(); i++)
        if (n.get(i) % 2 == 1)
            e.add(n.get(i));
    return e.toArray();
}
```

Java: List of ***even*** numbers using **IntStream**

```python
def filter_nums(max_val):
    nums = range(max_val)
    return [i for i in nums if i % 2 == 0]
```

Python: List of ***even*** numbers using list-comprehension

```python
def func(nums):
    if not nums:
        return nums
    elif nums[0] % 2 == 0:
        return [nums[0]] + func(nums[1:])
    else:
        return func(nums[1:])
```

Python: List of ***even*** numbers using recursion

Code-to-code Search - Language

```java
List<Integer> getOdds(int max) {
    List<Integer> odds = new ArrayList<>();
    for(int i = 0; i < max; i++)
        if (i % 2 == 1)
            odds.add(i);
    return odds;
}
```

Java: **for** loop to populate array of ***odd*** numbers

```java
Integer[] func(int x) {
    int[] n = IntStream.range(0, x).toArray();
    List<Integer> e = new ArrayList<>();
    for (int i=0; i<n.length(); i++)
        if (n.get(i) % 2 == 1)
            e.add(n.get(i));
    return e.toArray();
}
```

Java: List of ***even*** numbers using **IntStream**

```haskell
sift :: [Int] -> [Int]
sift [] = []
sift (x:xs) = if (x `mod` 2 == 0) then
                x: sift xs
              else
                sift xs

twoMultiples Int :: [Int]
twoMultiples n = sift [0..n-1]
```

Haskell: List of ***even*** numbers using recursion

```haskell
isEven :: Int -> Bool
isEven x = x `mod` 2 == 0

getEvens Int :: [Int]
getEvens n = filter (isEven x) [0..n]
```

Haskell: List of ***even*** numbers using chaining

```python
def filter_nums(max_val):
    nums = range(max_val)
    return [i for i in nums if i % 2 == 0]
```

Python: List of ***even*** numbers using list-comprehension

```python
def func(nums):
    if not nums:
        return nums
    elif nums[0] % 2 == 0:
        return [nums[0]] + func(nums[1:])
    else:
        return func(nums[1:])
```

Python: List of ***even*** numbers using recursion

Code-to-code Search - Behavior

```java
List<Integer> getOdds(int max) {
    List<Integer> odds = new ArrayList<>();
    for(int i = 0; i < max; i++)
        if (i % 2 == 1)
            odds.add(i);
    return odds;
}
```

Java: **for** loop to populate array of ***odd*** numbers

```haskell
sift :: [Int] -> [Int]
sift [] = []
sift (x:xs) = if (x `mod` 2 == 0) then
                 x: sift xs
              else
                 sift xs

twoMultiples Int :: [Int]
twoMultiples n = sift [0..n-1]
```

Haskell: List of ***even*** numbers using recursion

```haskell
isEven :: Int -> Bool
isEven x = x `mod` 2 == 0

getEvens Int :: [Int]
getEvens n = filter (isEven x) [0..n]
```

Haskell: List of ***even*** numbers using chaining

```java
Integer[] func(int x) {
    int[] n = IntStream.range(0, x).toArray();
    List<Integer> e = new ArrayList<>();
    for (int i=0; i<n.length(); i++)
        if (n.get(i) % 2 == 1)
            e.add(n.get(i));
    return e.toArray();
}
```

Java: List of ***even*** numbers using **IntStream**

```python
def filter_nums(max_val):
    nums = range(max_val)
    return [i for i in nums if i % 2 == 0]
```

Python: List of ***even*** numbers using list-comprehension

```python
def func(nums):
    if not nums:
        return nums
    elif nums[0] % 2 == 0:
        return [nums[0]] + func(nums[1:])
    else:
        return func(nums[1:])
```

Python: List of ***even*** numbers using recursion

Code-to-code Search - Structure

```java
List<Integer> getOdds(int max) {
    List<Integer> odds = new ArrayList<>();
    for(int i = 0; i < max; i++)
        if (i % 2 == 1)
            odds.add(i);
    return odds;
}
```

Java: **for** loop to populate array of ***odd*** numbers

```haskell
sift :: [Int] -> [Int]
sift [] = []
sift (x:xs) = if (x `mod` 2 == 0) then
                  x: sift xs
              else
                  sift xs

twoMultiples Int :: [Int]
twoMultiples n = sift [0..n-1]
```

Haskell: List of ***even*** numbers using recursion

```haskell
isEven :: Int -> Bool
isEven x = x `mod` 2 == 0

getEvens Int :: [Int]
getEvens n = filter (isEven x) [0..n]
```

Haskell: List of ***even*** numbers using chaining

```java
Integer[] func(int x) {
    int[] n = IntStream.range(0, x).toArray();
    List<Integer> e = new ArrayList<>();
    for (int i=0; i<n.length(); i++)
        if (n.get(i) % 2 == 1)
            e.add(n.get(i));
    return e.toArray();
}
```

Java: List of ***even*** numbers using **IntStream**

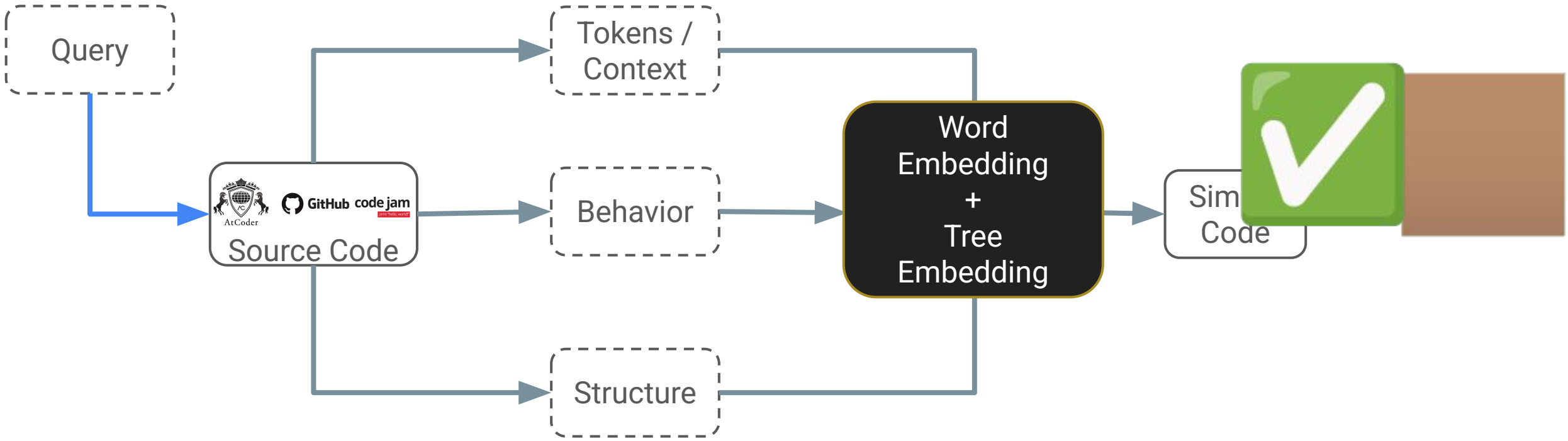
```python
def filter_nums(max_val):
    nums = range(max_val)
    return [i for i in nums if i % 2 == 0]
```

Python: List of ***even*** numbers using list-comprehension

```python
def func(nums):
    if not nums:
        return nums
    elif nums[0] % 2 == 0:
        return [nums[0]] + func(nums[1:])
    else:
        return func(nums[1:])
```

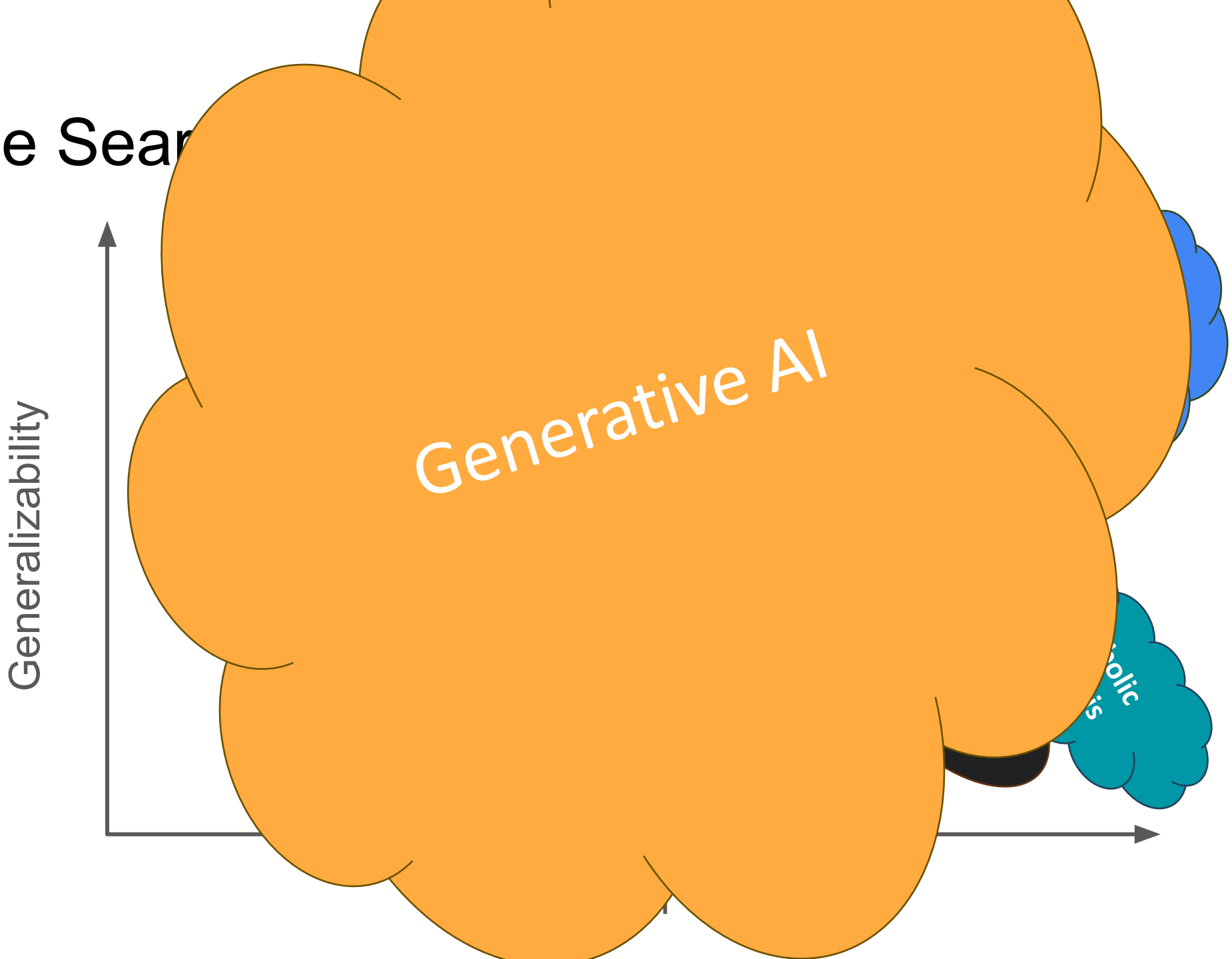
Python: List of ***even*** numbers using recursion

Code-to-code Search - In Practice

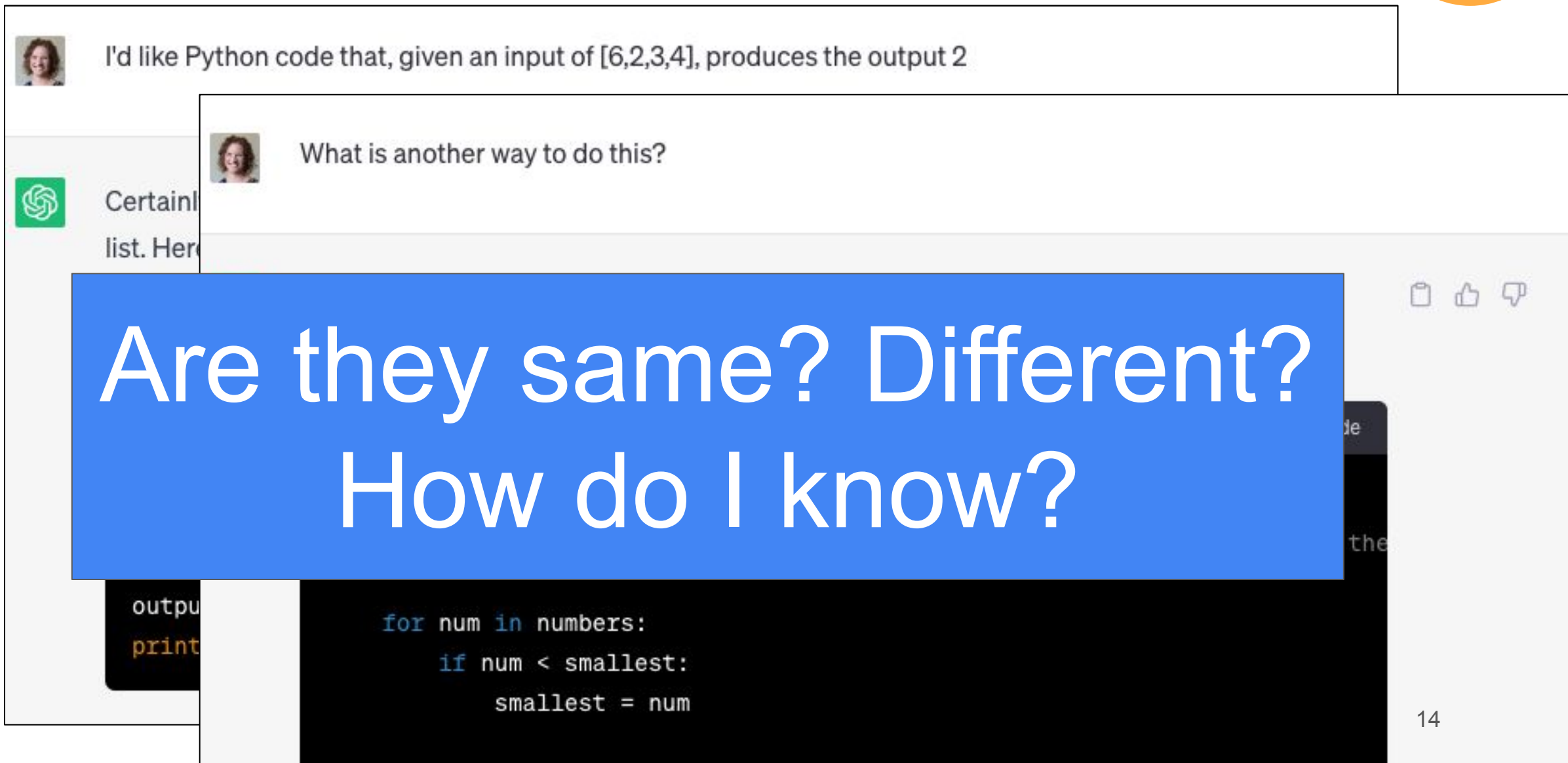


[ICSE 2020] [FSE 2021]

Code Sear



Looking Ahead…



Are they same? Different?
How do I know?

FIND IT

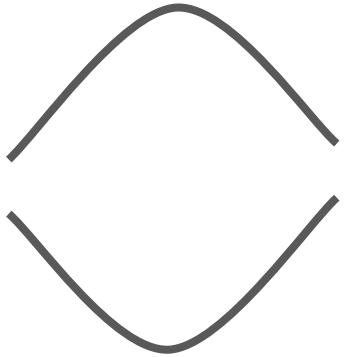
THINK IT

CHOOSE IT

Comparative Comprehension

*The cognitive activity of understanding how algorithms behave **relative to each other***

```python
def sumup(numbers):
  accumulator = 0
  for value in numbers:
    accumulator += value
  return accumulator
```



```python
def sumup(x):
  s = 0
  i = 0
  while i < len(x):
    s += i
    i += 1
  return s
```

Controlled Experiment

```java
public static boolean isAnagram(String str1,
                                String str2) {
  if (str1.length() != str2.length())
    return false;

  int[] count1 = new int[256];
  int[] count2 = new int[256];

  for (int i = 0; i < str1.length(); i++) {
    ++count1[str1.charAt(i)];
    ++count2[str2.charAt(i)];
  }

  for (int i = 0; i < 256; i++)
    if (count1[i] != count2[i])
      return false;

  return true;
}
```

```python
def isAnagram(s, t):
  hash1 = [0]*256
  hash2 = [0]*256

  for char in s:
    hash1[ord(char)] += 1
  for char in t:
    hash2[ord(char)] += 1

  return hash1 == hash2
```

- 4 independent dimensions of variation
  - Behavior (same or not)
  - Language (same or not)
  - Structures (similar AST or not)
  - Meaningful names (original or obfuscated)

Controlled Experiment

Thinkaloud Interviews

n=16

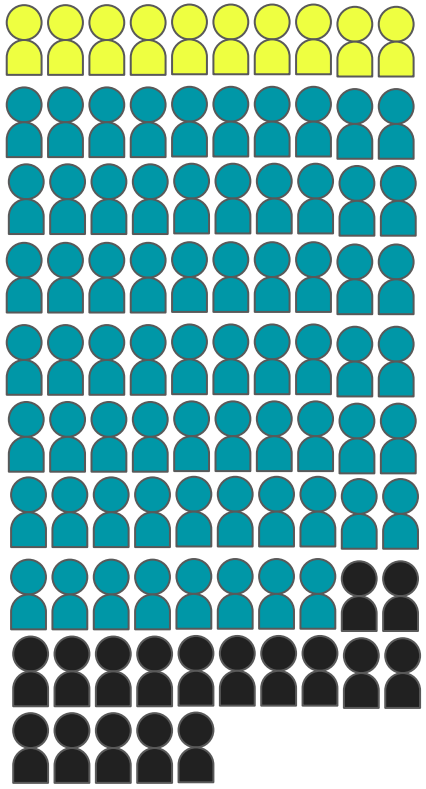
Undergraduate students

Graduate students

Professionals

Survey

n=95



Unknown

Graduate students

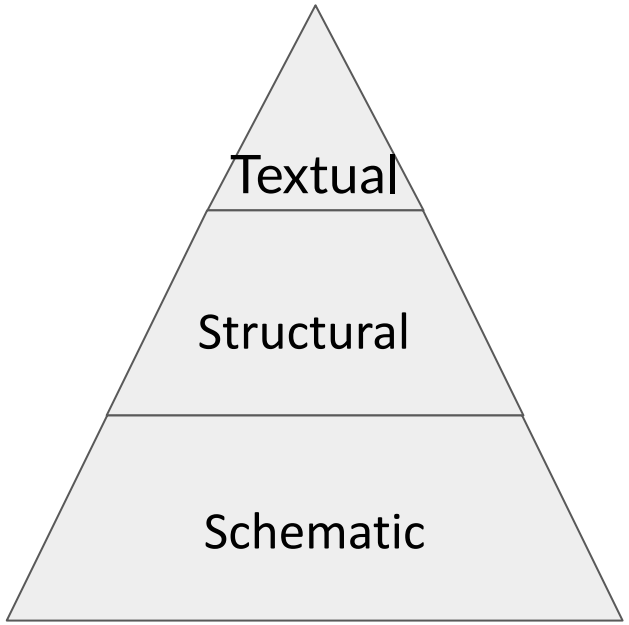
Professionals

Comparison Accuracy

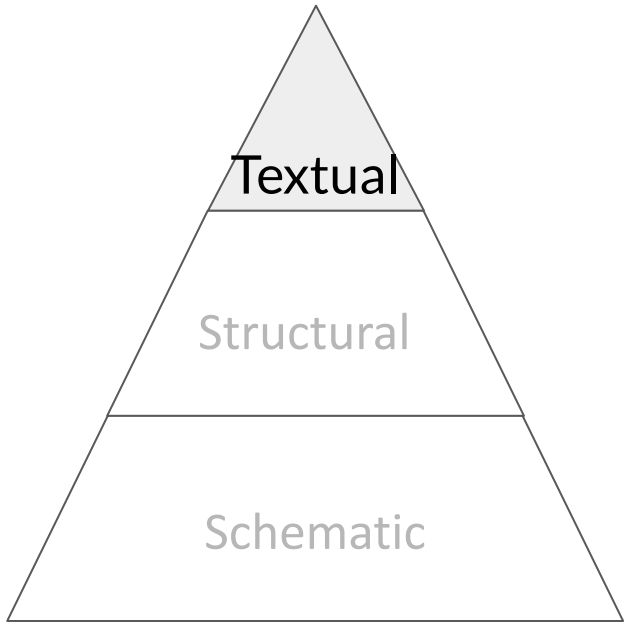
Overall correctness: 292 of 439 — 66.5%

	Correctness (%) for…		
	Similarity	**Dissimilarity**	
Clone Truth	85.3	46.7	***
Language	70.9	62.7	*
Structure	75.0	59.9	*
Names (Meaningful|Obf.)	66.8	66.2	

Comparison Strategies



Comparison Strategies

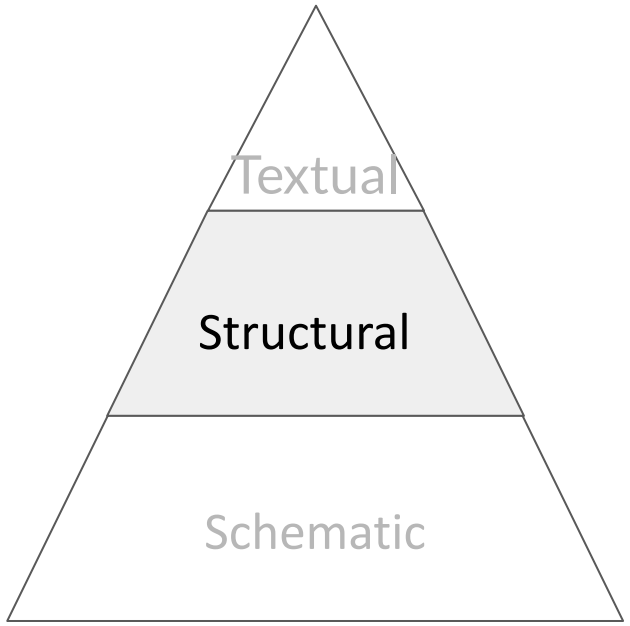



```python
def camel_case(string):
  a = list(string)
  for i in range(0, len(a)):
    if i==0 or a[i-1]=='_':
      a[i] = a[i].upper()
  return ''.join(a).replace('_','')
```

```python
def camel_case(string):
  a = list(string)
  for i in range(len(a)):
    if i==0 or a[i-1]=='_':
      a[i] = a[i].upper()
  return ''.join([c for c in a if c != "_"])
```

“I didn't even need to [understand the logic] because they were so similar.” - P4

Comparison Strategies

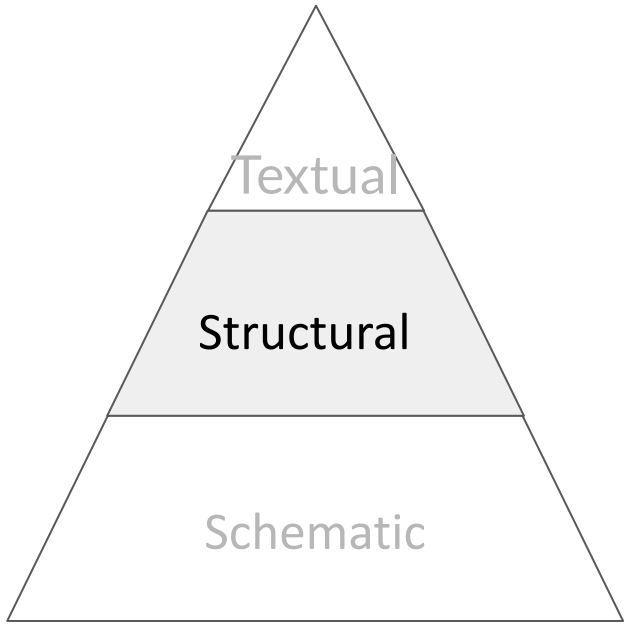



```java
public static String removeDuplicates(String orig)
{
  if (orig.length() == 0) return "";
  String result = "" + orig.charAt(0);
  for (int i = 1; i < orig.length(); i++)
    if (orig.charAt(i-1) != orig.charAt(i))
      result+=orig.charAt(i);
  return result;
}
```

```python
def removeadjacent(orig):
  if len(orig) == 0: return orig
  res = ""
  for i in range(1, len(orig)):
    if orig[i-1] != orig[i]:
      res += orig[i]
  return orig[0] + res
```

U4 on cross-language deduplicators

Comparison Strategies

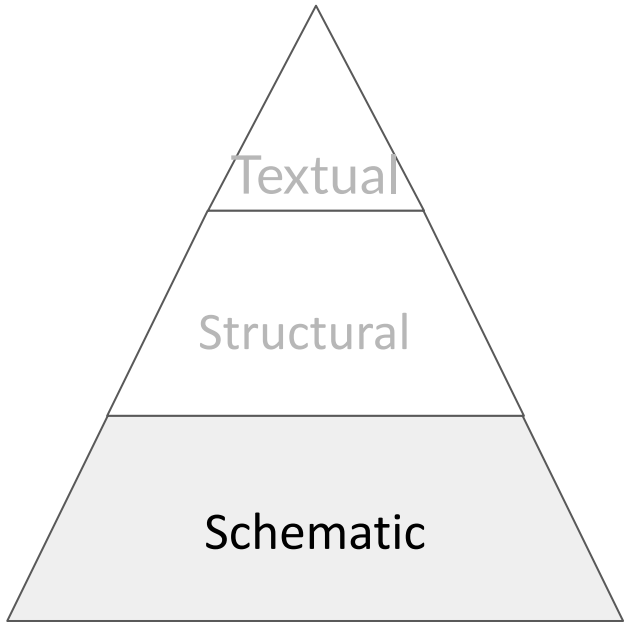



```java
public static String removeDuplicates(String orig)
{
  if (orig.length() == 0) return "";
  String result = "" + orig.charAt(0);
  for (int i = 1; i < orig.length(); i++)
    if (orig.charAt(i-1) != orig.charAt(i))
      result+=orig.charAt(i);
  return result;
}
```

```python
def removeadjacent(orig):
  if len(orig) == 0: return orig
  res = ""
  for i in range(1, len(orig)):
    if orig[i-1] != orig[i]:
      res += orig[i]
  return orig[0] + res
```

U4 on cross-language deduplicators

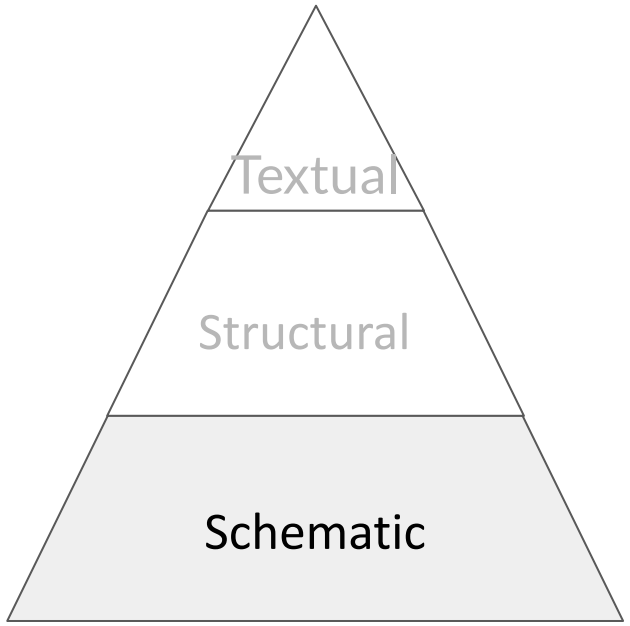
Comparison Strategies




```python
def to_camel_case(text):
  cap = True
  newText = ''
  for t in text:
    if t == '_':
      cap = True
      continue
    else:
      if cap == True:
        t = t.upper()
      newText = newText + t
      cap = False
  return newText
```

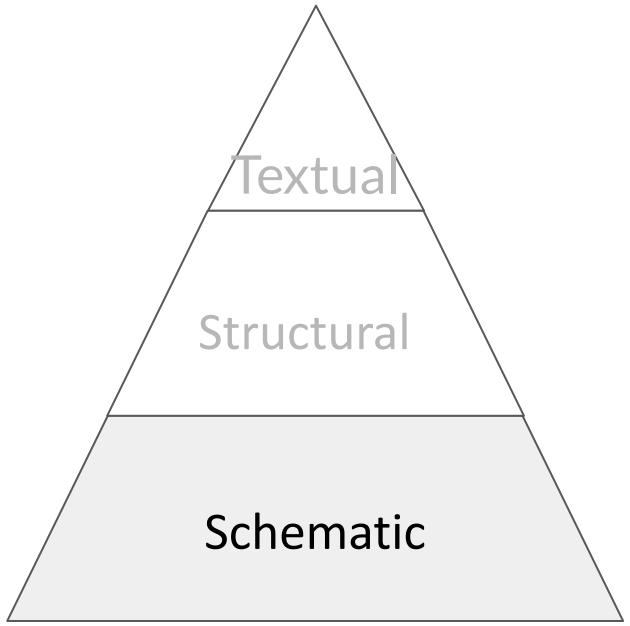
```python
def UnderscoreToCamelCase(under_score):
  segments = under_score.split('_')
  return segments[0] + ''.join([s[0].upper() + s[1:] \
    for s in segments[1:] if len(s) > 0])
```

Comparison Strategies




```
def UnderscoreToCamelCase(under_score):
  segments = under_score.split('_')
  return segments[0] + ''.join([s[0].upper() + s[1:] \
    for s in segments[1:] if len(s) > 0])
```

Comparison Strategies



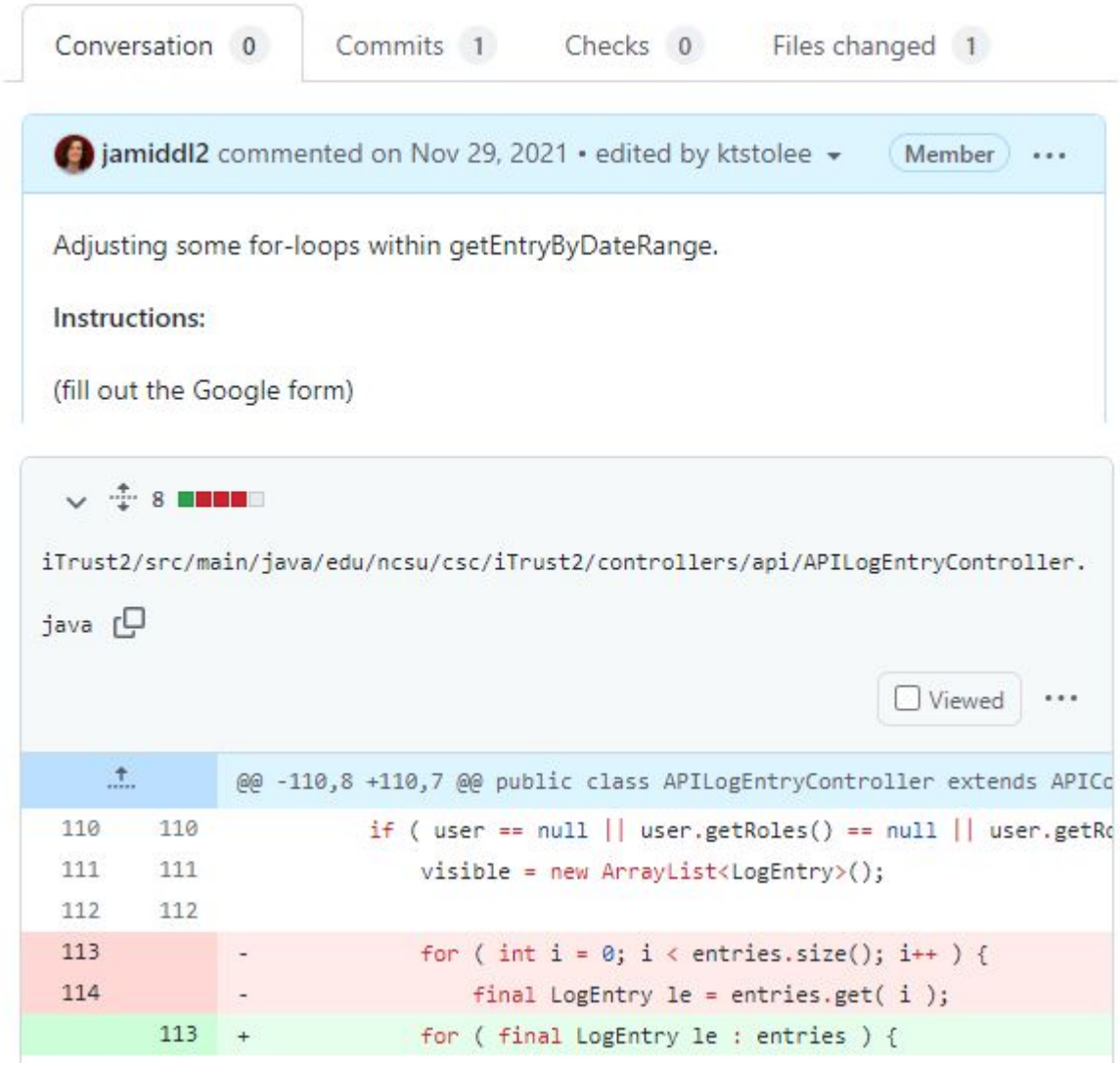

```python
def to_camel_case(text):
    cap = True
    newText = ''
    for t in text:
      if t == '_':
        cap = True
        continue
      else:
        if cap == True:
          t = t.upper()
        newText = newText + t
        cap = False
    return newText
```

What happens when comparative comprehension is done… on real code?

i.e., software engineering students reviewing code changes on GitHub in a code base they used in their class project.

it’s a step in the right direction.

Refactoring Review Study



RQ1: What **barriers** do student developers face when comprehending code changes?

RQ2: How **accurately** do student developers recognize behavioral impact in code review tasks?

Study Context

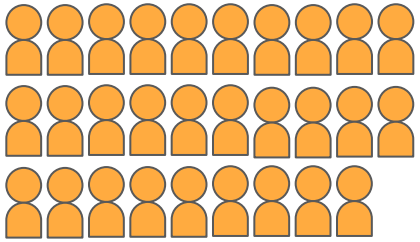
Junior-level undergraduate students

Interviews (20 minutes)

On Zoom

Before the tasks

n=29

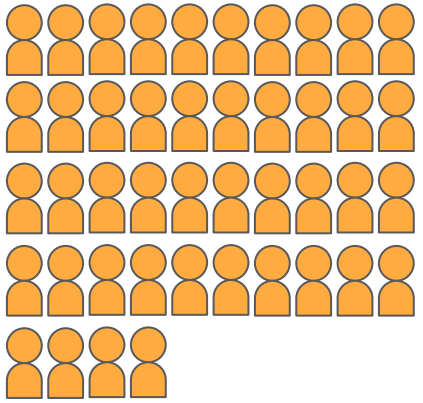


8 Tasks (40+ minutes)

In-class study on refactoring review using GitHub

Familiar code base

n=44



Interviews

- Prior to class activity
- 10-20 minutes
- Semi-structured

(1) **Background questions (repeated in post-activity survey)**:
   (a) Are you familiar with the concepts of code refactoring? → Please define in your own words.
   (b) And code review, especially with a team? → Please define in your own words.
   (c) What have been your experiences in receiving code reviews?
   (d) What experiences have you had outside of class?

(2) **Review Techniques questions:**
   (a) When doing code review on new or changed behavior, what tools and techniques did you have to determine behavior?
   (b) Are those techniques good enough, or do you wish you had a better way?
   (c) Do you typically get to see both versions of the code at the same time?

(3) **Quality & Refactoring Questions**
   (a) Have you made suggestions during code review to improve code quality without changing the overall behavior?
   (b) How do you define code quality in these situations?
   (c) What techniques did you have to determine quality?
   (d) What techniques did you have to determine if behavior has been maintained?
   (e) Do those techniques typically work as intended and in a timely manner, or could they be better?

(4) **Speculative Questions**
   (a) If you had access to any information or tool you'd like, what would be your ideal way of examining, comparing, and reviewing code?
   (b) What are your biggest sources of frustration?
   (c) What are other times you compare code, beyond review?

Tasks

6 refactoring review tasks were retained for analysis (some true refactorings, some non-refactorings)

1. `for` loop → `for each` loop
2. loop → pipeline
3. consolidate conditional + extract variable
4. consolidate conditional + extract and move function
5. Replace magic literal + `Steam.collect.size` → `steam.count`
6. Extract function + slide statement

Task link here: https://github.ncsu.edu/engr-csc326-fall2021/csc326-ref-activity-000/pull/30

Is this code change a refactoring (i.e. it does not change the external behavior)?

- ◯ Yes, it is a refactoring.
- ◯ No, it changes the code's behavior.
- ◯ I don't know.

What impact does this code change have?

If it is a refactoring, does the refactoring improve the legibility, maintainability, or something else? If it is not a refactoring, what behavior does this pull request change? If you do not know, what feature of the code are you unsure about?

Your answer

What tools or strategies did you use to investigate the differences in the code?

Explain in a comment what strategies you used in comparison. Did you use an IDE? Did you use the unified or split view in GitHub? Did you run the test suite? All of the above? None of the above?

Your answer

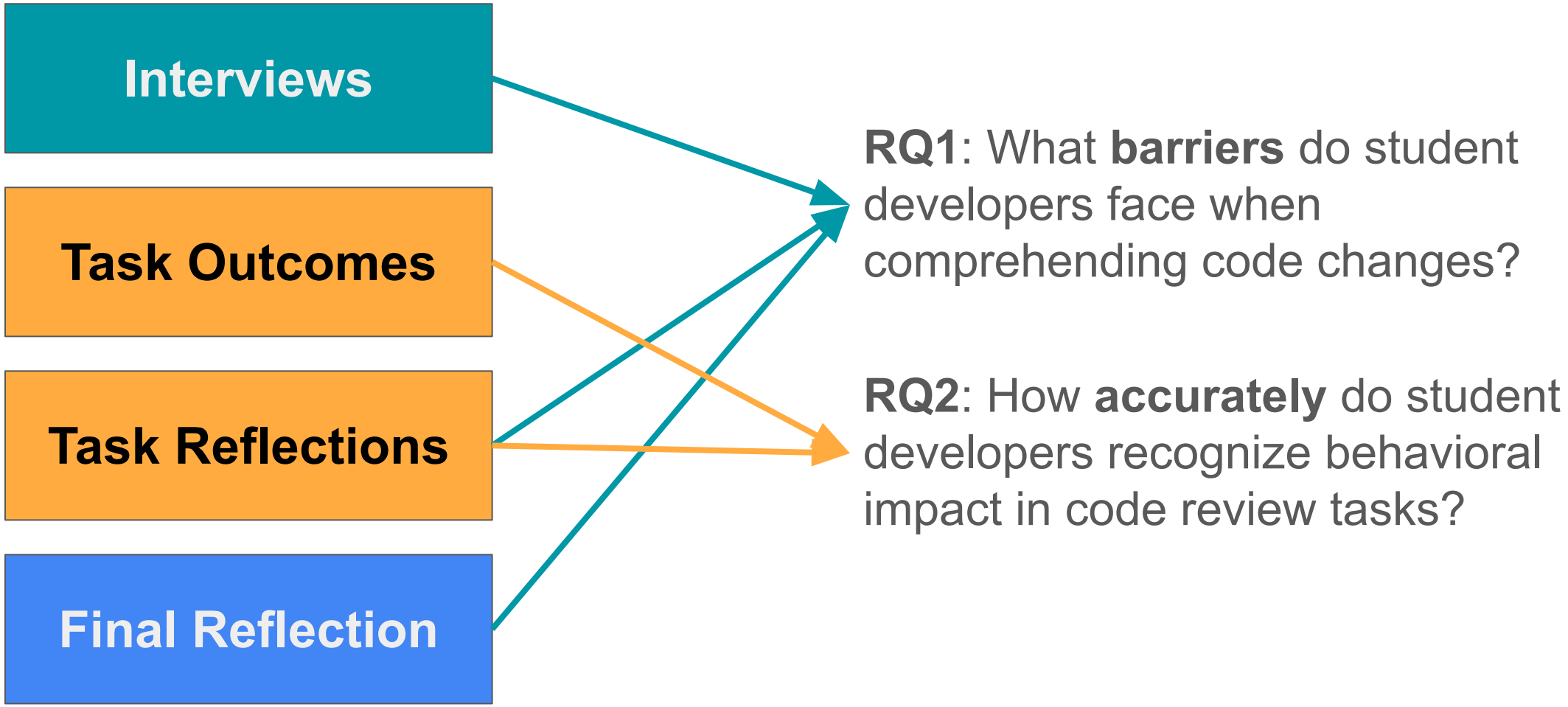
Was there anything difficult about comparing this code?

Your answer

Post-Task Reflection

(1) What was difficult about performing the code review in this study?
(2) What was easy about performing the code review in this study?
(3) What would have helped you perform the code review in this study more effectively?
(4) How many years of programming experience do you have?
(5) Do you have experience in professional software environments? With code review? With refactoring?
(6) What is your gender identity? *[male/female/non-binary/prefer not to disclose]*

Refactoring Review Study



true refactorings
non-refactorings

Results RQ2 - Accuracy

	Responses	# Refactoring	#Non-Refactoring
`for` loop → `for each` loop	42	35 (83%)	**7 (17%)**
loop -> pipeline	43	**34 (79%)**	2 (5%)
consolidate conditional + extract variable	44	38 (86%)	**5 (11%)**
consolidate conditional + extract and move function	44	**31 (70%)**	10 (23%)
Replace magic literal + `Steam.collect.size → steam.count`	38	**27 (71%)**	4 (11%)
Extract function + slide statement	31	22 (71%)	**2 (6%)**

Overall Accuracy: 106 / 242 = 43.8%

Results RQ1 - Barriers

Title	Description	Interview n=29	Activity n=44	Reflection n=44
	Context Barriers			
Limited Time	Insufficient time to perform the task to the developer's satisfaction.	2 (7%)	4 (9%)	10 (23%)
Social Friction	Dysfunctions or a lack of response from other developers.	12 (41%)	0 (0%)	2 (5%)
Self-Doubt	Difficulty because of lack of experience or lack of self-confidence.	1 (3%)	3 (7%)	1 (2%)
	All Context Barriers	13 (45%)	6 (14%)	13 (30%)
	Tool Barriers			
Lack of Tests	Insufficient automatic verification of the codebase.	4 (14%)	3 (7%)	4 (9%)
Limited or Misaligned View	Cannot focus on all relevant code at once; limited screen space.	8 (28%)	17 (39%)	7 (16%)
Toolchain Issues	Dysfunctions in coordination of tools.	7 (24%)	1 (2%)	1 (2%)
	All Tool Barriers	13 (45%)	18 (41%)	12 (27%)
	Code Barriers			
Large Scope	Large volume of code to comprehend.	8 (28%)	11 (25%)	8 (18%)
Unfamiliar Code	Code is unfamiliar or uses unfamiliar features.	1 (3%)	21 (48%)	10 (23%)
Comprehension	Code is difficult to understand.	8 (28%)	21 (48%)	6 (14%)
	All Code Barriers	12 (41%)	32 (73%)	23 (52%)
	Comparative Comprehension Barriers			
Unclear Motivation	The developer does not know why code was written or changed.	10 (34%)	3 (7%)	2 (5%)
Deep Changes	New version of code looks very different.	0 (0%)	7 (16%)	2 (5%)
Merge Conflicts	Dysfunctions in deciding the authoritative versions.	3 (10%)	0 (0%)	0 (0%)
Delta Comprehension	The changes between code versions are difficult to understand.	1 (3%)	15 (34%)	4 (9%)
	All Comparative Comprehension Barriers	13 (45%)	18 (41%)	8 (18%)

Results RQ1 - Barriers

Title	Description	Interview n=29	Activity n=44	Reflection n=44
	Context Barriers			
Limited Time	Insufficient time to perform the task to the developer's satisfaction.	2 (7%)	4 (9%)	10 (23%)
Social Friction	Dysfunctions or a lack of response from other developers.	12 (41%)	0 (0%)	2 (5%)
Self-Doubt	Difficulty because of lack of experience or lack of self-confidence.	1 (3%)	3 (7%)	1 (2%)
	All Context Barriers	13 (45%)	6 (14%)	13 (30%)
	Tool Barriers			
Lack of Tests	Insufficient automatic verification of the codebase.	4 (14%)	3 (7%)	4 (9%)
Limited or Misaligned View	Cannot focus on all relevant code at once; limited screen space.	8 (28%)	17 (39%)	7 (16%)
Toolchain Issues	Dysfunctions in coordination of tools.	7 (24%)	1 (2%)	1 (2%)
	All Tool Barriers	13 (45%)	18 (41%)	12 (27%)
	Code Barriers			
Large Scope	Large volume of code to comprehend.	8 (28%)	11 (25%)	8 (18%)
Unfamiliar Code	Code is unfamiliar or uses unfamiliar features.	1 (3%)	21 (48%)	10 (23%)
Comprehension	Code is difficult to understand.	8 (28%)	21 (48%)	6 (14%)
	All Code Barriers	12 (41%)	32 (73%)	23 (52%)
	Comparative Comprehension Barriers			
Unclear Motivation	The developer does not know why code was written or changed.	10 (34%)	3 (7%)	2 (5%)
Deep Changes	New version of code looks very different.	0 (0%)	7 (16%)	2 (5%)
Merge Conflicts	Dysfunctions in deciding the authoritative versions.	3 (10%)	0 (0%)	0 (0%)
Delta Comprehension	The changes between code versions are difficult to understand.	1 (3%)	15 (34%)	4 (9%)
	All Comparative Comprehension Barriers	13 (45%)	18 (41%)	8 (18%)

[Code] Comprehension

P25: *“The new boolean statements were somewhat **difficult to parse**”*

P29: “*Yes, **trying to understand certain methods** and what the code was accomplishing [was difficult].*”

50 iTrust2/src/main/java/edu/ncsu/csc/iTrust2/forms/ICDCode

```
174       -         else if ( !isOphthalmology.equals( other.isOphthal
      137 +         } else if ( obj == null || getClass() != obj.getCl
175   138               return false;
176   139           }
177       -         return true;
      140 +
      141 +         final ICDCodeForm other = (ICDCodeForm) obj;
      142 +         final boolean sameCode = code != null && code.equa
      143 +         final boolean sameDescription = description != nul
                  description.equals(other.description);
      144 +         final boolean sameId = id != null && id.equals(oth
      145 +         final boolean sameOpthalmology = isOphthalmology !
      146 +                             isOphthalmology.equals(other.isOph
      147 +         return sameCode && sameDescription && sameId && sa
```

[Code] Unfamiliar Code

P8: *"This was a bit more difficult as I am **not experienced in using array streams**"*

P32: "*I **did not know what the code's purpose was**, so I had to look at the entire file...Only looking at the change lines was difficult.*"

13 iTrust2/src/main/java/edu/ncsu/csc/iTrust2/models

```
102  103          *          could be found
103  104          */
104  105         public static BloodType parse ( final String typ
105       -          for ( final BloodType type : values() ) {
106       -              if ( type.getName().equals( typeStr ) )
107       -                  return type;
108       -              }
109       -          }
110       -          return NotSpecified;
     106  +          return Arrays.stream(values())
     107  +                       .filter(type -> type.getName
     108  +                       .findFirst()
     109  +                       .orElse(NotSpecified);
111  110          }
112       - }
     111  + }
```

[Code] Large Scope

P27: *“Given that there were a **huge change**, at first it was overwhelming to read all of the code.”*

P12: “*This [pull request] could be improved by **separating smaller changes into more commits**, but that doesn't appear possible in this example.*”

Results RQ1 - Barriers

Title	Description	Interview n=29	Activity n=44	Reflection n=44
	Context Barriers			
Limited Time	Insufficient time to perform the task to the developer's satisfaction.	2 (7%)	4 (9%)	10 (23%)
Social Friction	Dysfunctions or a lack of response from other developers.	12 (41%)	0 (0%)	2 (5%)
Self-Doubt	Difficulty because of lack of experience or lack of self-confidence.	1 (3%)	3 (7%)	1 (2%)
	All Context Barriers	13 (45%)	6 (14%)	13 (30%)
	Tool Barriers			
Lack of Tests	Insufficient automatic verification of the codebase.	4 (14%)	3 (7%)	4 (9%)
Limited or Misaligned View	Cannot focus on all relevant code at once; limited screen space.	8 (28%)	17 (39%)	7 (16%)
Toolchain Issues	Dysfunctions in coordination of tools.	7 (24%)	1 (2%)	1 (2%)
	All Tool Barriers	13 (45%)	18 (41%)	12 (27%)
	Code Barriers			
Large Scope	Large volume of code to comprehend.	8 (28%)	11 (25%)	8 (18%)
Unfamiliar Code	Code is unfamiliar or uses unfamiliar features.	1 (3%)	21 (48%)	10 (23%)
Comprehension	Code is difficult to understand.	8 (28%)	21 (48%)	6 (14%)
	All Code Barriers	12 (41%)	32 (73%)	23 (52%)
	Comparative Comprehension Barriers			
Unclear Motivation	The developer does not know why code was written or changed.	10 (34%)	3 (7%)	2 (5%)
Deep Changes	New version of code looks very different.	0 (0%)	7 (16%)	2 (5%)
Merge Conflicts	Dysfunctions in deciding the authoritative versions.	3 (10%)	0 (0%)	0 (0%)
Delta Comprehension	The changes between code versions are difficult to understand.	1 (3%)	15 (34%)	4 (9%)
	All Comparative Comprehension Barriers	13 (45%)	18 (41%)	8 (18%)

[Context] Social Friction

P12: “*I don't know I **don't like giving people negative feedback** when it's when it's like really strongly negative.*”

P43: “***some group members, … always think that they're right**.*”

P44: *“If I told someone to review code, they could write a comment and **there's no way to check and see if they've actually looked through the code**”*

Results RQ1 - Barriers

Title	Description	Interview n=29	Activity n=44	Reflection n=44
	Context Barriers			
Limited Time	Insufficient time to perform the task to the developer's satisfaction.	2 (7%)	4 (9%)	10 (23%)
Social Friction	Dysfunctions or a lack of response from other developers.	12 (41%)	0 (0%)	2 (5%)
Self-Doubt	Difficulty because of lack of experience or lack of self-confidence.	1 (3%)	3 (7%)	1 (2%)
	All Context Barriers	13 (45%)	6 (14%)	13 (30%)
	Tool Barriers			
Lack of Tests	Insufficient automatic verification of the codebase.	4 (14%)	3 (7%)	4 (9%)
Limited or Misaligned View	Cannot focus on all relevant code at once; limited screen space.	8 (28%)	17 (39%)	7 (16%)
Toolchain Issues	Dysfunctions in coordination of tools.	7 (24%)	1 (2%)	1 (2%)
	All Tool Barriers	13 (45%)	18 (41%)	12 (27%)
	Code Barriers			
Large Scope	Large volume of code to comprehend.	8 (28%)	11 (25%)	8 (18%)
Unfamiliar Code	Code is unfamiliar or uses unfamiliar features.	1 (3%)	21 (48%)	10 (23%)
Comprehension	Code is difficult to understand.	8 (28%)	21 (48%)	6 (14%)
	All Code Barriers	12 (41%)	32 (73%)	23 (52%)
	Comparative Comprehension Barriers			
Unclear Motivation	The developer does not know why code was written or changed.	10 (34%)	3 (7%)	2 (5%)
Deep Changes	New version of code looks very different.	0 (0%)	7 (16%)	2 (5%)
Merge Conflicts	Dysfunctions in deciding the authoritative versions.	3 (10%)	0 (0%)	0 (0%)
Delta Comprehension	The changes between code versions are difficult to understand.	1 (3%)	15 (34%)	4 (9%)
	All Comparative Comprehension Barriers	13 (45%)	18 (41%)	8 (18%)

[Comparative Comprehension] Unclear Motivation

P23: "*When I **don't have [change] documentation, that definitely slows down the process** of me being able to understand and interpret what their code is doing.*

P2: "*The code change was a little confusing as **I didn't see a compelling "code smell"**.*"

```
137       -       if ( ovf.getId() != null ) {
138       -           ov.setId( Long.parseLong( ovf.getId() ) );
      137 +       String id = ovf.getId();
      138 +       if ( id != null ) {
      139 +           ov.setId( Long.parseLong( id ) );
139   140         }
140   141
141       -       final ZonedDateTime visitDate = ZonedDateTime.parse( ovf.getDate() );
142       -       ov.setDate( visitDate );
      142 +       ov.setDate( ZonedDateTime.parse( ovf.getDate() ) );
```

Results RQ1 - Barriers

Title	Description	Interview n=29	Activity n=44	Reflection n=44
	Context Barriers			
Limited Time	Insufficient time to perform the task to the developer's satisfaction.	2 (7%)	4 (9%)	10 (23%)
Social Friction	Dysfunctions or a lack of response from other developers.	12 (41%)	0 (0%)	2 (5%)
Self-Doubt	Difficulty because of lack of experience or lack of self-confidence.	1 (3%)	3 (7%)	1 (2%)
	All Context Barriers	13 (45%)	6 (14%)	13 (30%)
	Tool Barriers			
Lack of Tests	Insufficient automatic verification of the codebase.	4 (14%)	3 (7%)	4 (9%)
Limited or Misaligned View	Cannot focus on all relevant code at once; limited screen space.	8 (28%)	17 (39%)	7 (16%)
Toolchain Issues	Dysfunctions in coordination of tools.	7 (24%)	1 (2%)	1 (2%)
	All Tool Barriers	13 (45%)	18 (41%)	12 (27%)
	Code Barriers			
Large Scope	Large volume of code to comprehend.	8 (28%)	11 (25%)	8 (18%)
Unfamiliar Code	Code is unfamiliar or uses unfamiliar features.	1 (3%)	21 (48%)	10 (23%)
Comprehension	Code is difficult to understand.	8 (28%)	21 (48%)	6 (14%)
	All Code Barriers	12 (41%)	32 (73%)	23 (52%)
	Comparative Comprehension Barriers			
Unclear Motivation	The developer does not know why code was written or changed.	10 (34%)	3 (7%)	2 (5%)
Deep Changes	New version of code looks very different.	0 (0%)	7 (16%)	2 (5%)
Merge Conflicts	Dysfunctions in deciding the authoritative versions.	3 (10%)	0 (0%)	0 (0%)
Delta Comprehension	The changes between code versions are difficult to understand.	1 (3%)	15 (34%)	4 (9%)
	All Comparative Comprehension Barriers	13 (45%)	18 (41%)	8 (18%)

[Tool] Limited or Misaligned Views

P24: "*The changes were on **two different files, which made it a bit trickier to compare** them.*"

P11: "*it was little difficult to understand since **the changed codes are scattered all around** two different codes.*"

Implications

- Social friction is likely a bigger issue than what we observed
  - Industry has seen this, too
- Support is needed to help with comprehension
  - Code summarization for single code?
  - Diffs of code summaries?
  - Behavioral diffing?
  - Test case generation to demonstrate differences?
  - … I’m just speculating here

Come see us at ICSE 2024!

Barriers for Students During Code Change Comprehension

Justin Middleton
Department of Computer Science
North Carolina State University
USA
jamiddl2@ncsu.edu

John-Paul Ore
Department of Computer Science
North Carolina State University
USA
jwore@ncsu.edu

Kathryn T. Stolee
Department of Computer Science
North Carolina State University
USA
ktstolee@ncsu.edu

ABSTRACT

Modern code review (MCR) is a key practice for many software engineering organizations, so undergraduate software engineering courses often teach some form of it to prepare students. However, research on MCR describes how many its professional implementations can fail, to say nothing on how these barriers manifest under students' particular contexts. To uncover barriers students face when evaluating code changes during review, we combine interviews and surveys with an observational study. In a junior-level software engineering course, we first interviewed 29 undergraduate students about their experiences in code review. Next, we performed an observational study that presented 44 students from the same course with eight code change comprehension activities. These activities provided students with pull requests of potential refactorings in a familiar code base, collecting feedback on accuracy

ACM Reference Format:
Justin Middleton, John-Paul Ore, and Kathryn T. Stolee. 2024. Barriers for Students During Code Change Comprehension. In *2024 IEEE/ACM 46th International Conference on Software Engineering (ICSE '24), April 14–20, 2024, Lisbon, Portugal.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3597503.3639227

1 INTRODUCTION

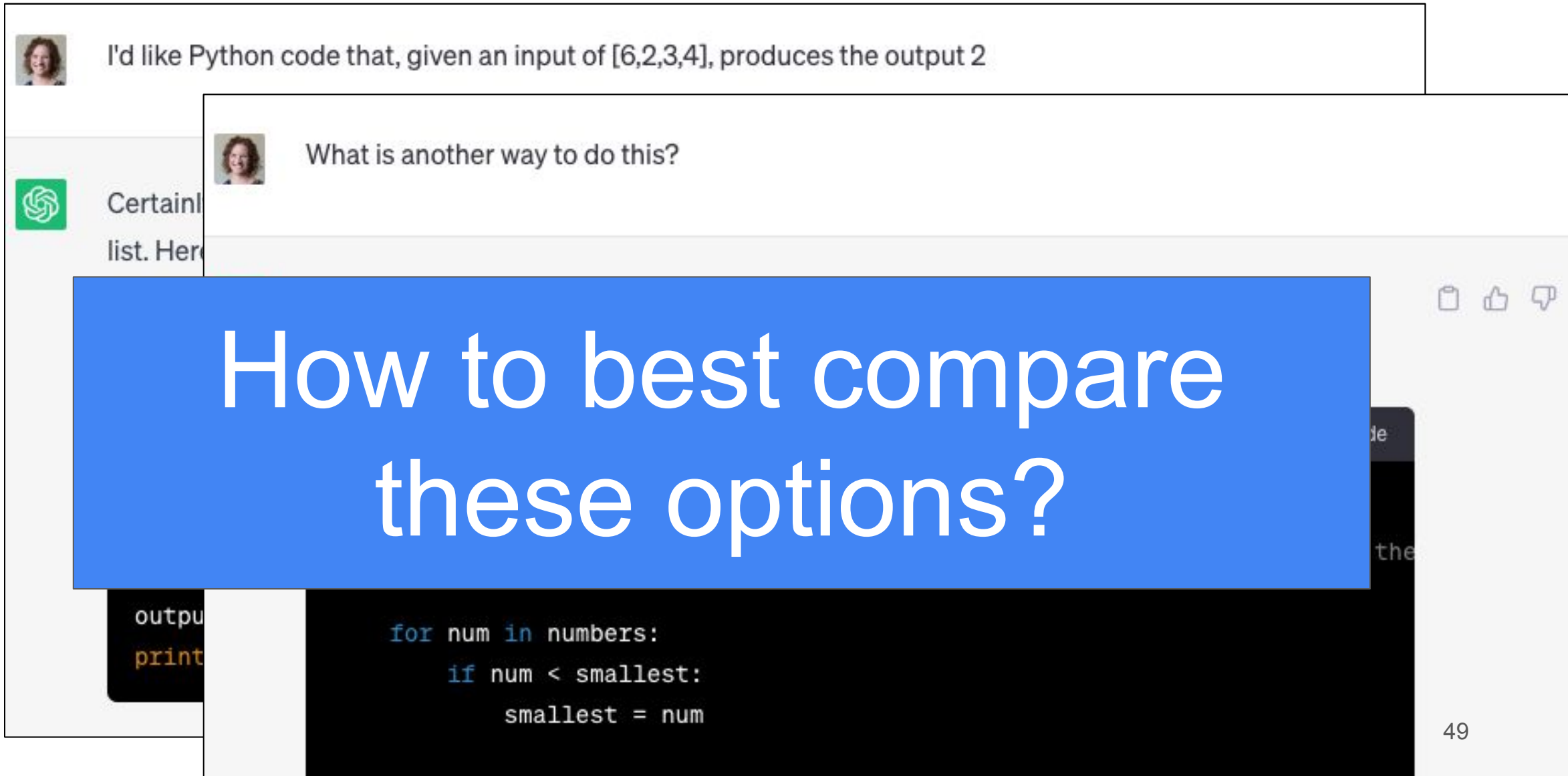
Code review is a useful and popular software engineering practice wherein team members manually inspect each other's new code to verify that it meets expectations before integrating it into the official product [4]. Not only does code review improve quality by catching bugs early [41], but it also promotes organizational

FIND IT

THINK IT

CHOOSE IT

Looking Ahead…



How to best compare these options?

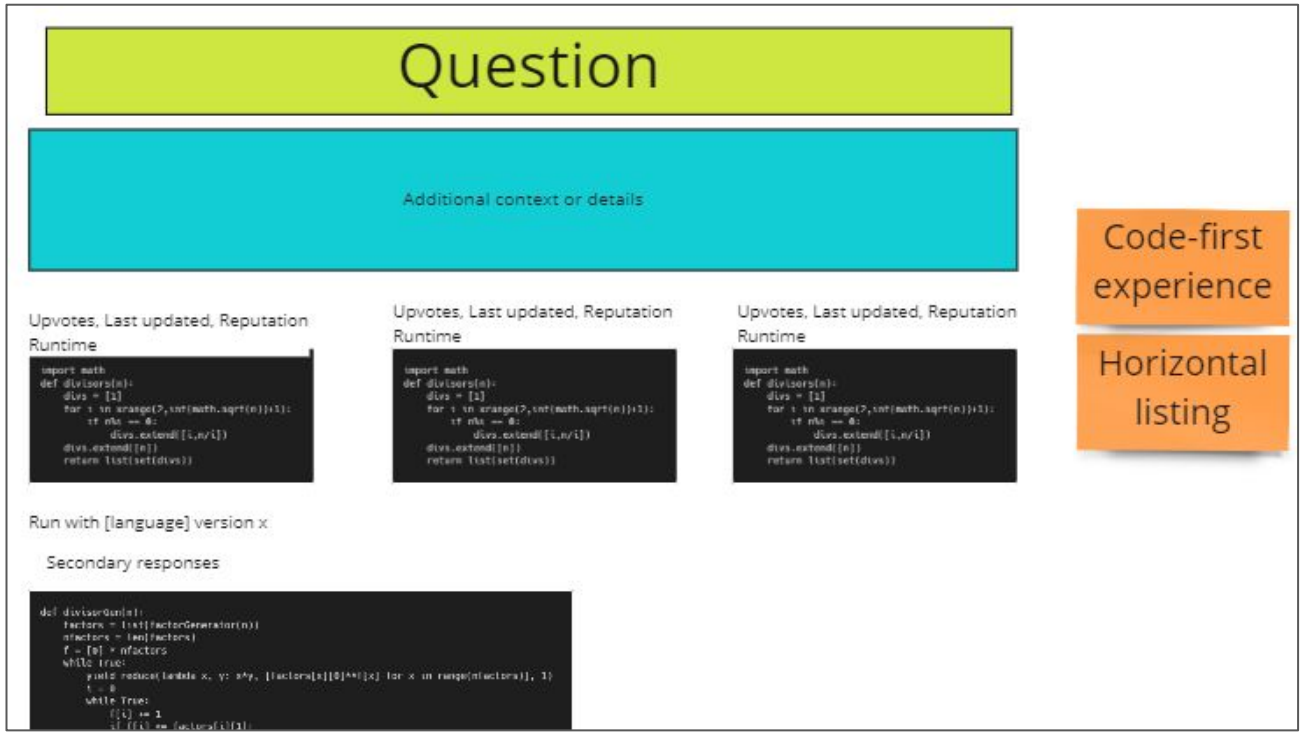
Participatory Design - an HCI technique

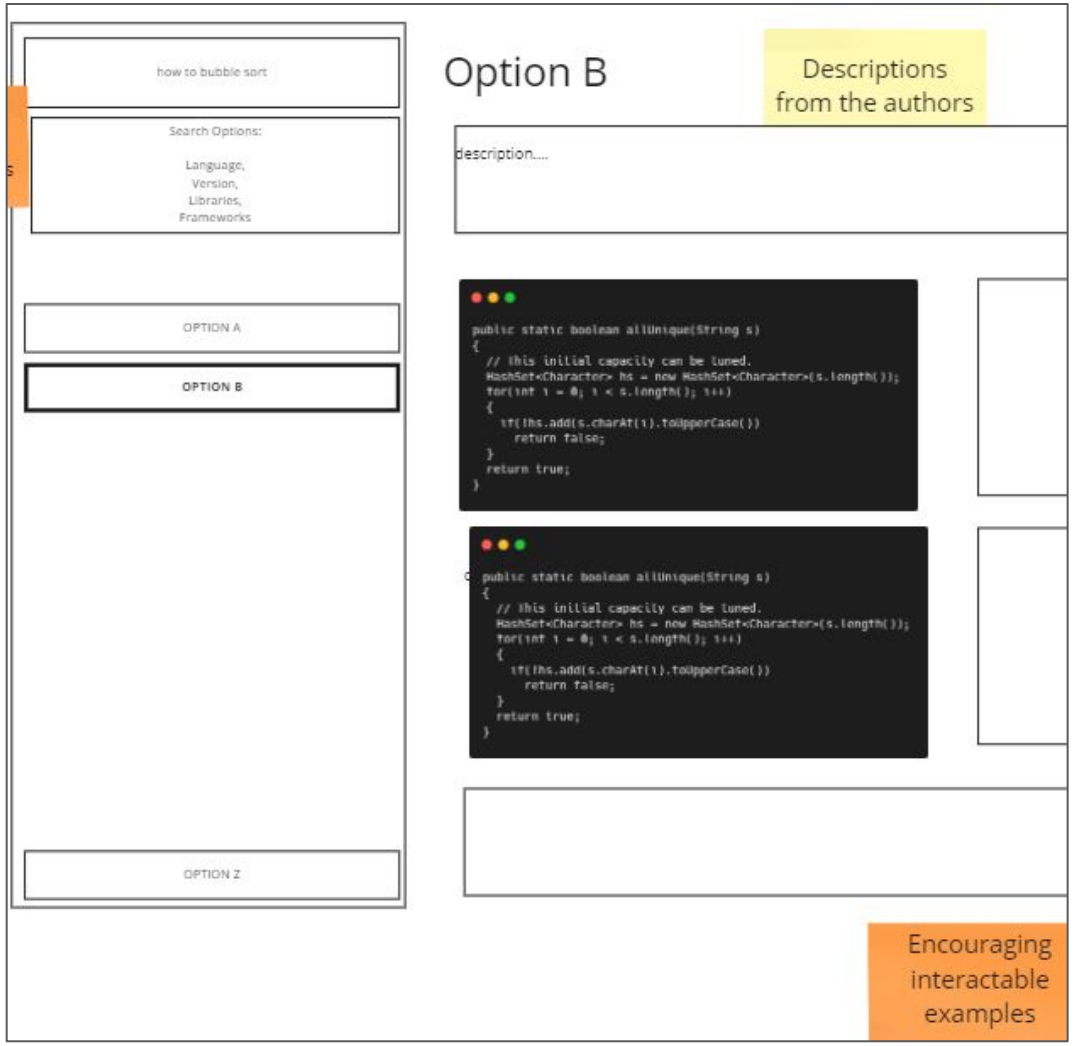
RQ1: What interface features do developers want to navigate multiple similar snippets?

RQ2: What interface arrangements do developers want to navigate multiple similar snippets?

Participant	Position	Years Exp. Programming	Years Exp. Prof. Programming	Years Exp. Prof. Designing	Languages	Assignments Learning	Assignments Searching	Assignments Recommending
P01	Graduate Student	5	2		Python, Java	✓		✓
P02	Graduate Student	6	1	1	Python, Java	✓	✓	
P03	Graduate Student	6	2		Python, Java		✓	✓
P04	Graduate Student	11	5	4	Python, Java	✓	✓	
P05	Graduate Student	9	2		Python, JavaScript	✓		✓
P06	Software Engineer	12	7		Python, Java		✓	✓
P07	Software Engineer	5	1		Python, Java	✓		✓
P08	Data Engineer	25	23	15	JavaScript, C#		✓	✓
P09	Software Engineer	11	7		Python, Java	✓		✓
P10	Data Scientist	10	5		Python		✓	✓
P11	Software Engineer	2	1		JavaScript	✓	✓	

Participatory Design





Code Comprehenion + Behavioral Diffing?

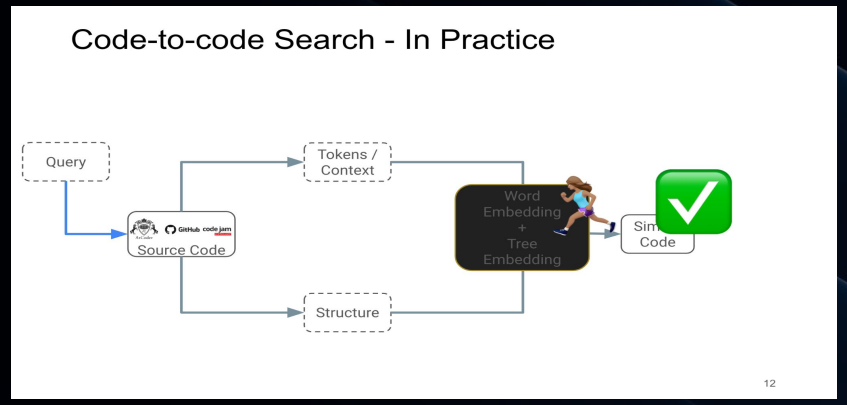
How does it help with comparative comprehension?

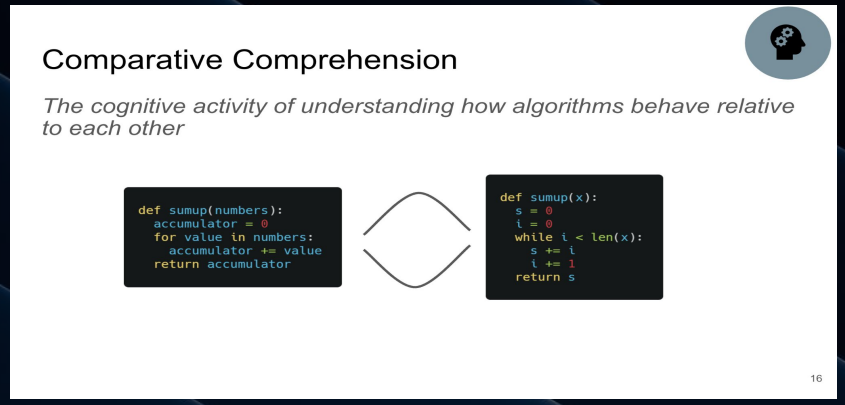
(Hey grad students! I believe this would be a straightforward project. Want to collaborate?)

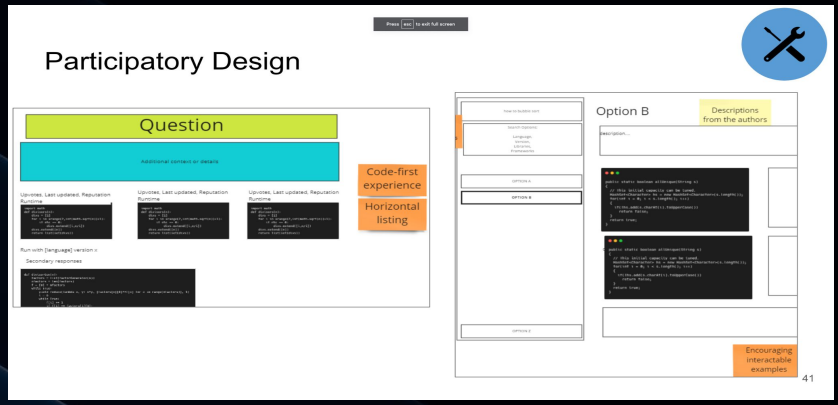
Augmenting Diffs With Runtime Information

Khashayar Etemadi, Aman Sharma, Fernanda Madeiral, Martin Monperrus

Abstract—Source code diffs are used on a daily basis as part of code review, inspection, and auditing. To facilitate understanding, they are typically accompanied by explanations that describe the essence of what is changed in the program. As manually crafting high-quality explanations is a cumbersome task, researchers have proposed automatic techniques to generate code diff explanations. Existing explanation generation methods solely focus on static analysis, i.e., they do not take advantage of runtime information to explain code changes. In this paper, we propose COLLECTOR-SAHAB, a novel tool that augments code diffs with runtime difference information. COLLECTOR-SAHAB compares the program states of the original (old) and patched (new) versions of a program to find unique variable values. Then, COLLECTOR-SAHAB adds this novel runtime information to the source code diff as shown, for instance, in code reviewing systems. As an evaluation, we run COLLECTOR-SAHAB on 584 code diffs for Defects4J bugs and find it successfully augments the code diff for 95% (555/584) of them. We also perform a user study and ask eight participants to score the augmented







FIND IT THINK IT CHOOSE IT

Teamwork makes it happen.



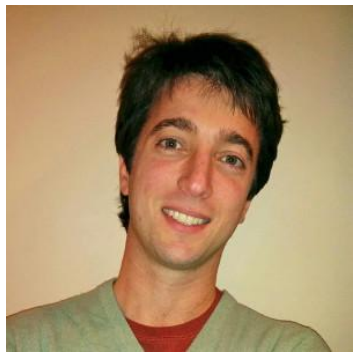
















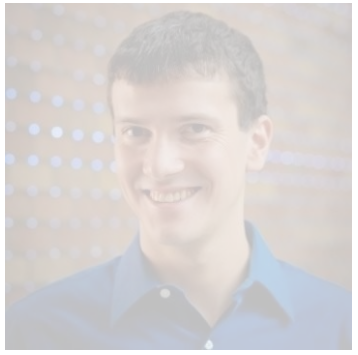
…. And more!

Teamwork makes it happen.

















…. And more!

Thanks!

ktstolee@ncsu.edu

ICSE 2024: labels vs. explanations

		Q2		
	Labeled as ↓	Says Behavior Changes	Says Behavior Does Not Change	Total
Q1	Refactoring	**1**	248	249
	Non-Refactoring	23	**12**	35
	Total	24	260	284