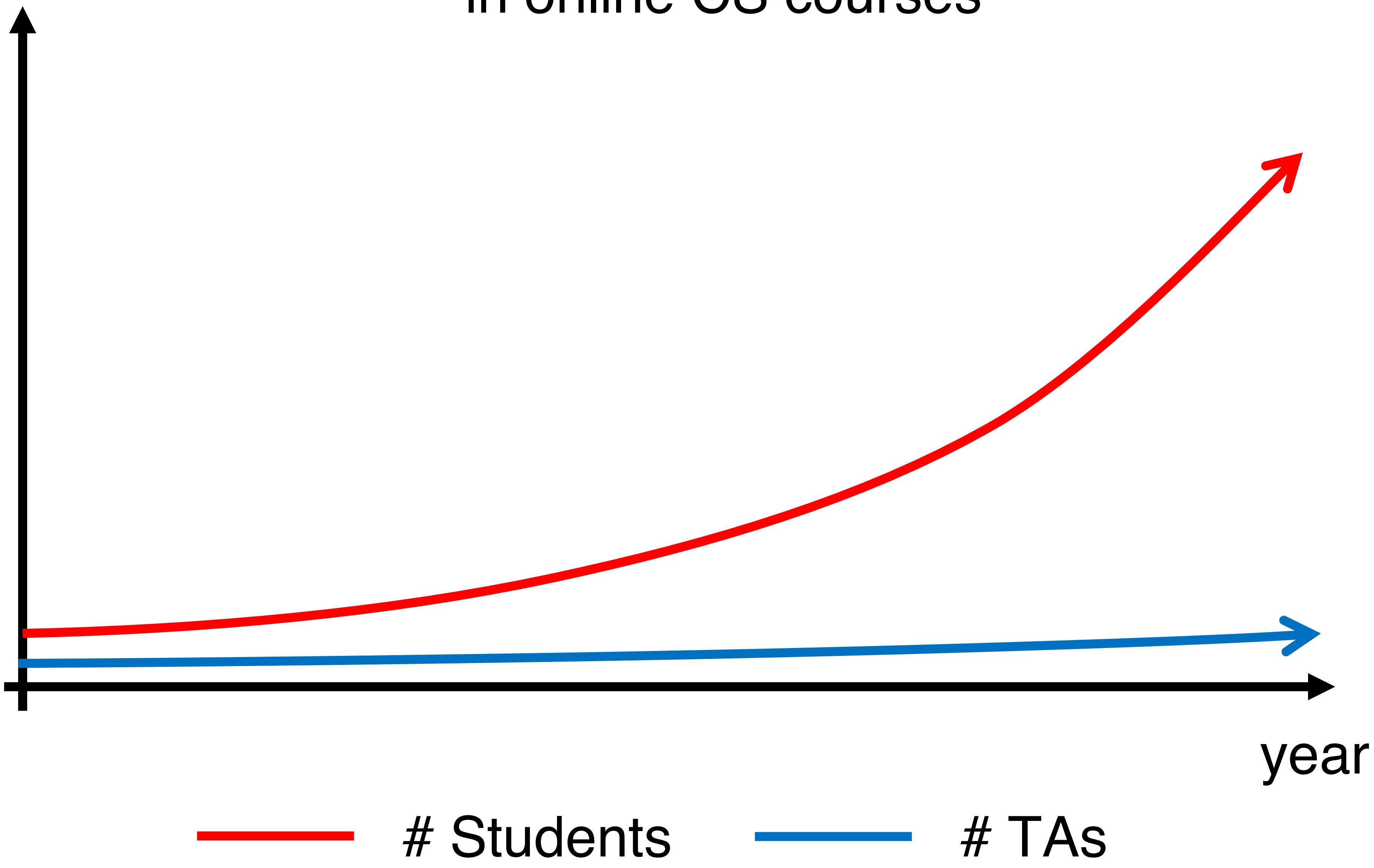
Eliph: Effective Visualization of Code History for Peer Assessment in Programming Education

Jungkook Park (School of Computing, KAIST)
Yeong Hoon Park (School of Computing, KAIST)
Suin Kim (School of Computing, KAIST)
Alice Oh (School of Computing, KAIST)

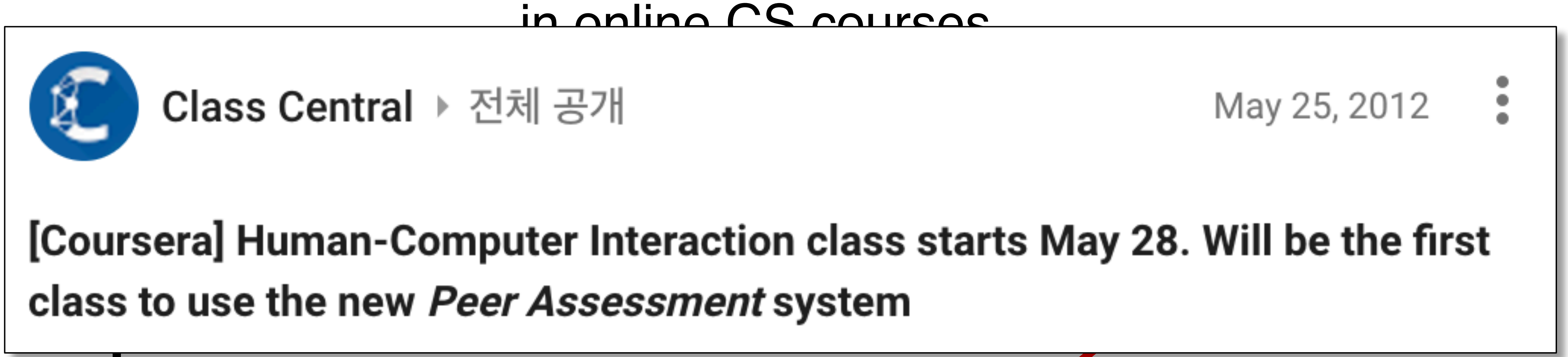
Rapid Growth of Online CS Courses

Numbers of students and TAs
in online CS courses



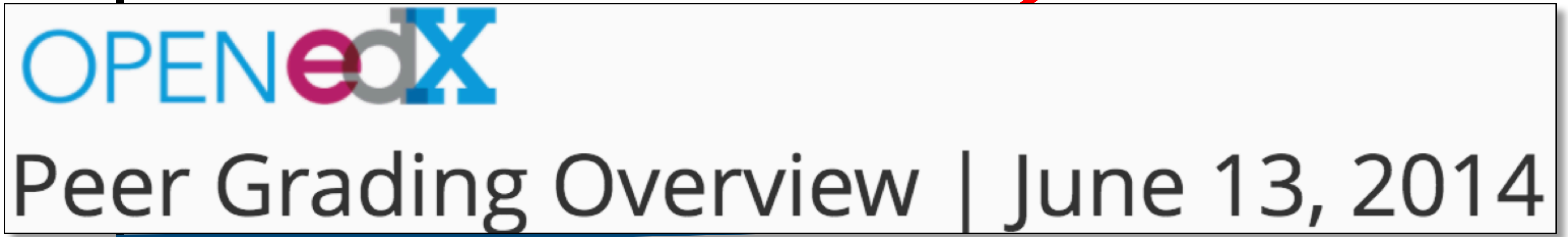
Rapid Growth of Online CS Courses

Numbers of students and TAs
in online CS courses



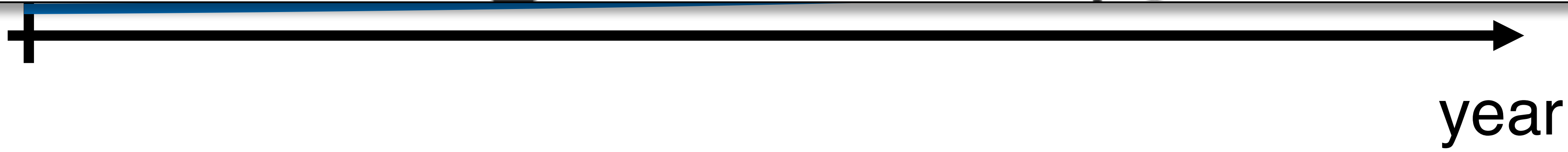
Class Central ▸ 전체 공개 May 25, 2012

[Coursera] Human-Computer Interaction class starts May 28. Will be the first class to use the new *Peer Assessment* system



OPENedX

Peer Grading Overview | June 13, 2014

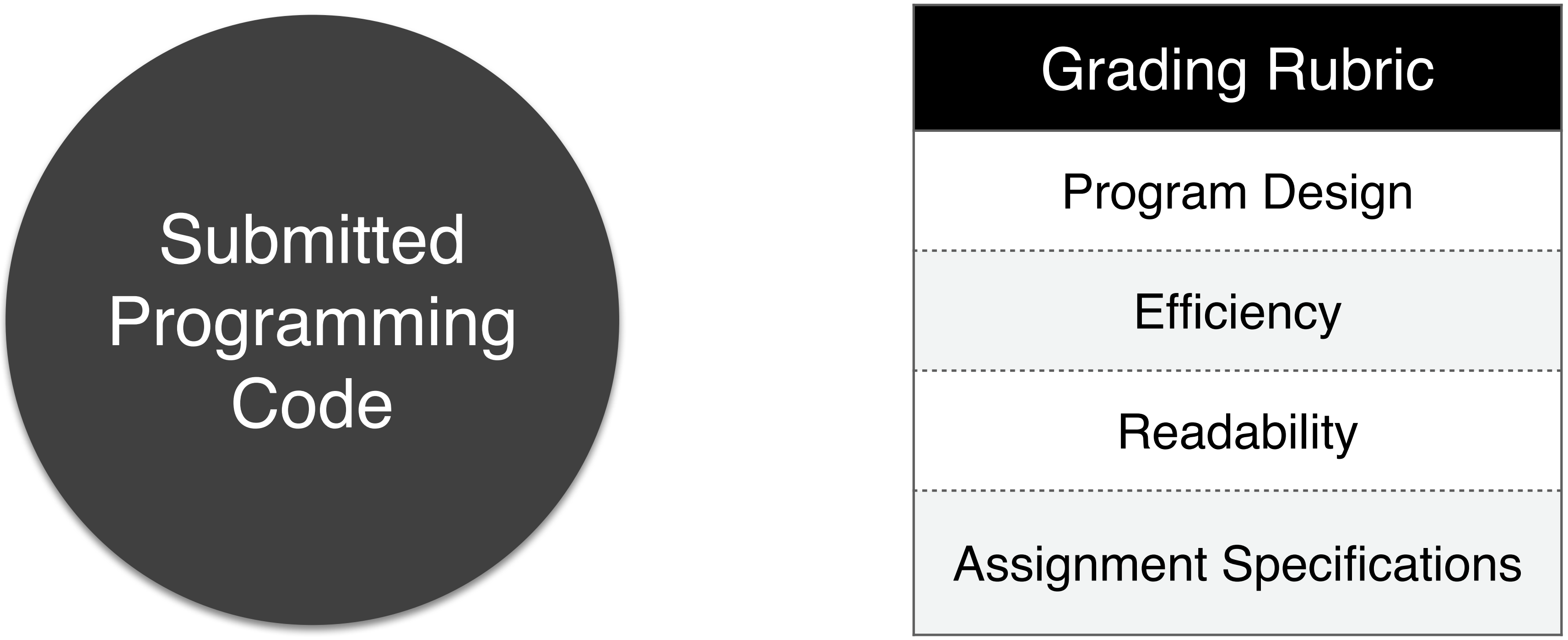


— # Students — # TAs

Difficulties of Peer Assessment in CS

Peer assessment involves both

“**understanding other’s work**” and “giving a proper mark”



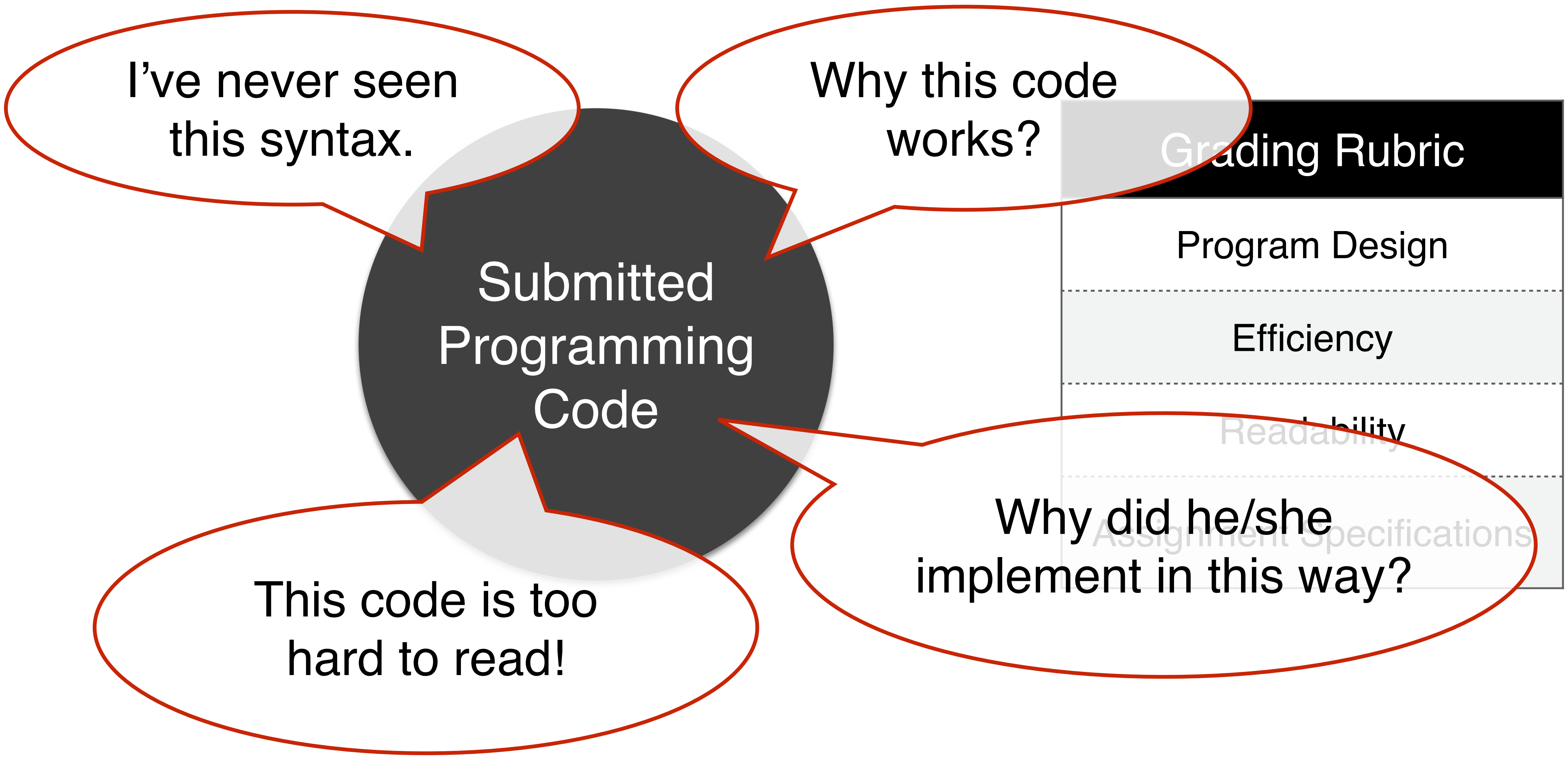
Submitted Programming Code

Grading Rubric
Program Design
Efficiency
Readability
Assignment Specifications

Difficulties of Peer Assessment in CS

Peer assessment involves both

"**understanding other's work**" and "giving a proper mark"

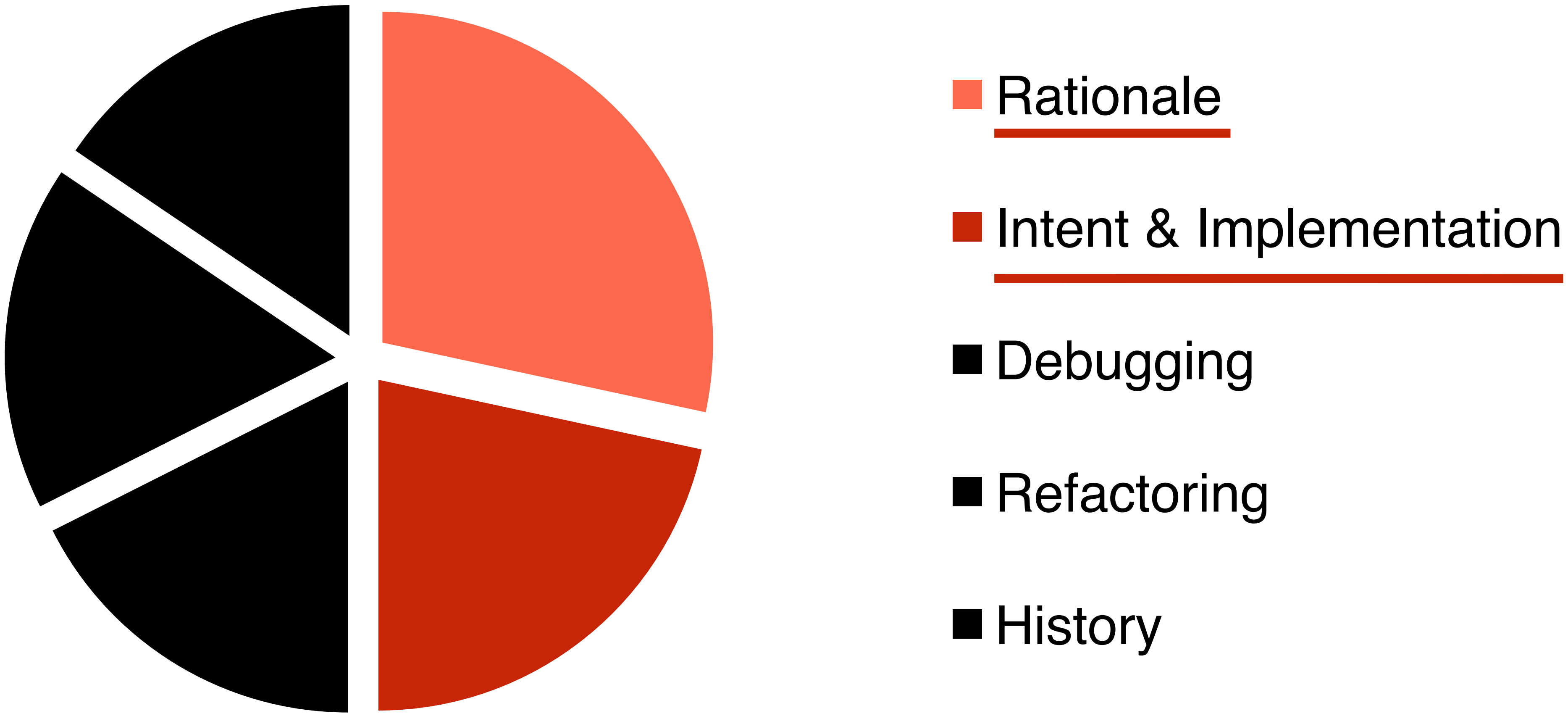

I've never seen this syntax.
Why this code works?
Submitted Programming Code
This code is too hard to read!
Why did he/she implement in this way?
Grading Rubric
Program Design
Efficiency
Readability
Assignment Specifications

Difficulties of Understanding Other’s Code

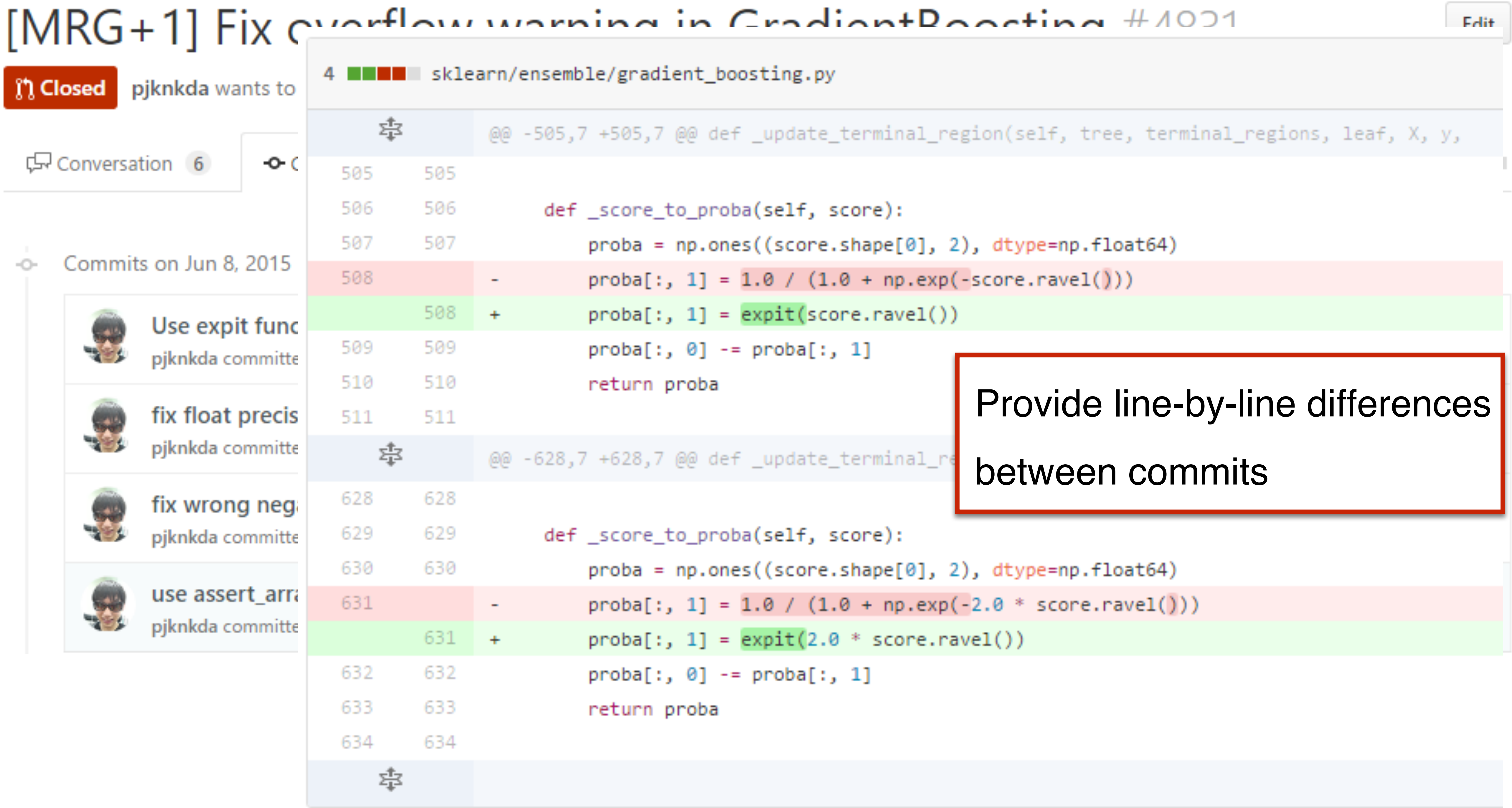
Even for skilled programmers, it is difficult to infer

the intentions of the code author by merely reading the code

What is the most difficult question if you are supposed to answer by reading other’s code?



Common Practice in Open Source Community



[MRG+1] Fix overflow warning in GradientBoosting #4921

Edit

Closed pjknkda wants to

Conversation 6

Commits on Jun 8, 2015

Use expit func
pjknkda committe

fix float precis
pjknkda committe

fix wrong neg
pjknkda committe

use assert_arr
pjknkda committe

4 sklearn/ensemble/gradient_boosting.py

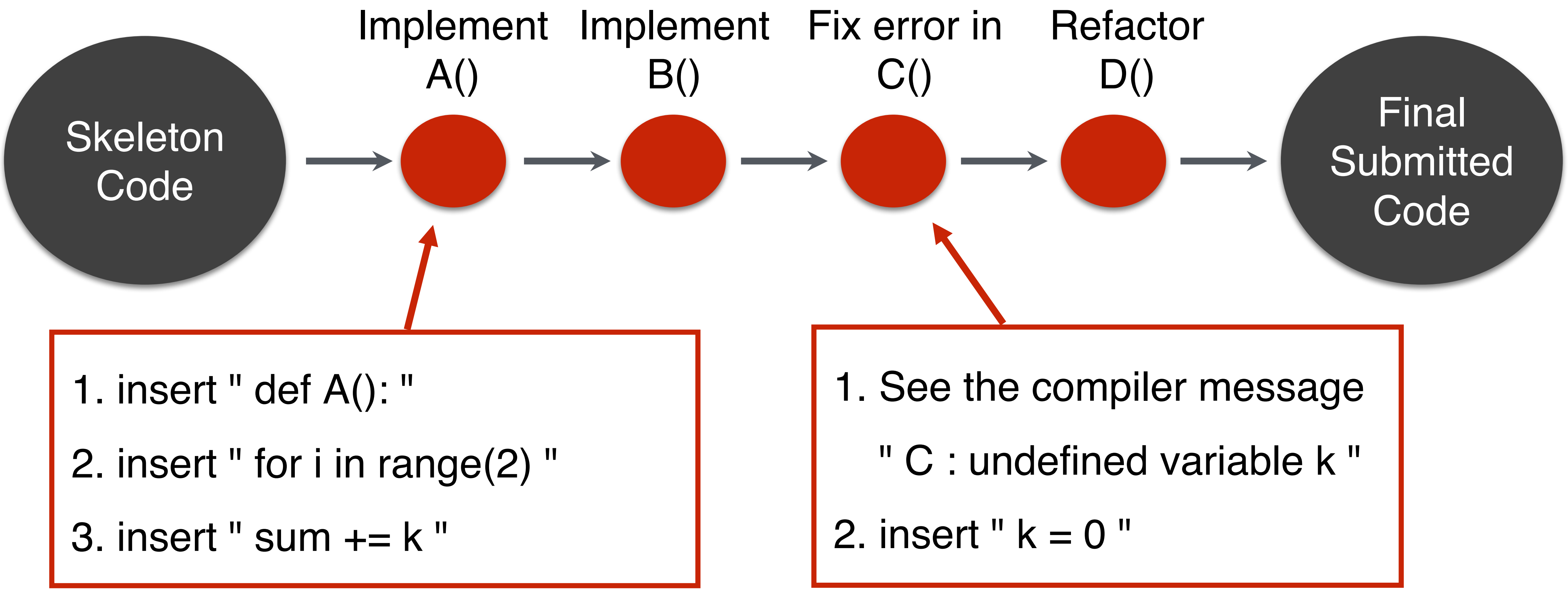
```
@@ -505,7 +505,7 @@ def _update_terminal_region(self, tree, terminal_regions, leaf, X, y,

     def _score_to_proba(self, score):
         proba = np.ones((score.shape[0], 2), dtype=np.float64)
-        proba[:, 1] = 1.0 / (1.0 + np.exp(-score.ravel()))
+        proba[:, 1] = expit(score.ravel())
         proba[:, 0] -= proba[:, 1]
         return proba

@@ -628,7 +628,7 @@ def _update_terminal_re

     def _score_to_proba(self, score):
         proba = np.ones((score.shape[0], 2), dtype=np.float64)
-        proba[:, 1] = 1.0 / (1.0 + np.exp(-2.0 * score.ravel()))
+        proba[:, 1] = expit(2.0 * score.ravel())
         proba[:, 0] -= proba[:, 1]
         return proba

```

Provide line-by-line differences between commits

Proposed Approach

Skeleton Code
Final Submitted Code

Proposed Approach



Skeleton Code

Implement A()

Implement B()

Fix error in C()

Refactor D()

Final Submitted Code

1. insert " def A(): "
2. insert " for i in range(2) "
3. insert " sum += k "

1. See the compiler message

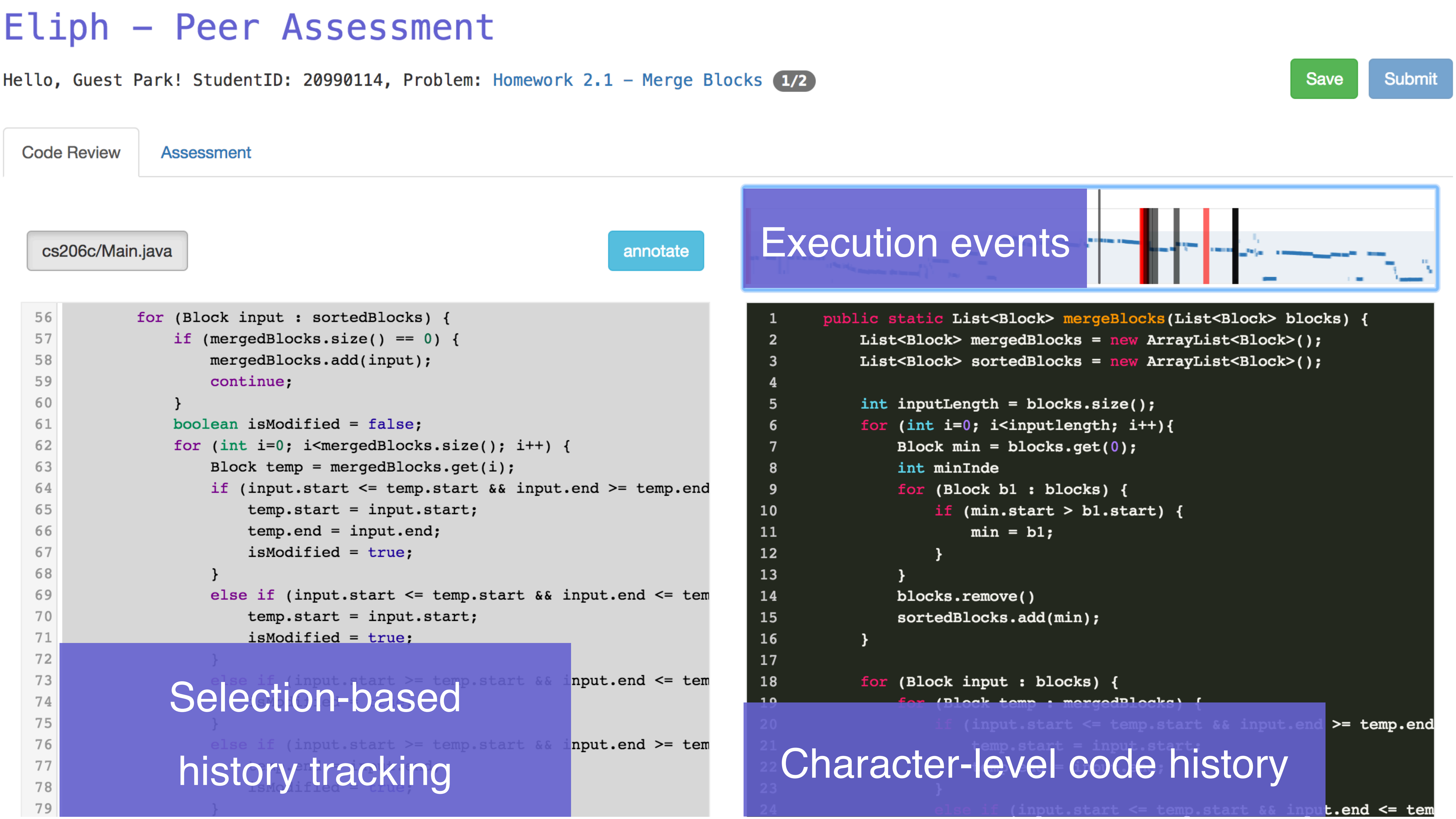
   " C : undefined variable k "

2. insert " k = 0 "

Eliph

A web-based peer assessment system
for CS education with code history visualization

Eliph



Hello, Guest Park! StudentID: 20990114, Problem: Homework 2.1 – Merge Blocks 1/2

```
        for (Block input : sortedBlocks) {
            if (mergedBlocks.size() == 0) {
                mergedBlocks.add(input);
                continue;
            }
            boolean isModified = false;
            for (int i=0; i<mergedBlocks.size(); i++) {
                Block temp = mergedBlocks.get(i);
                if (input.start <= temp.start && input.end >= temp.end
                    temp.start = input.start;
                    temp.end = input.end;
                    isModified = true;
                }
                else if (input.start <= temp.start && input.end <= tem
                    temp.start = input.start;
                    isModified = true;
                }
                else if (input.start >= temp.start && input.end <= tem
                    isModified = true;
                }
                else if (input.start >= temp.start && input.end >= tem
                    temp.end = input.end;
                    isModified = true;
                }
```

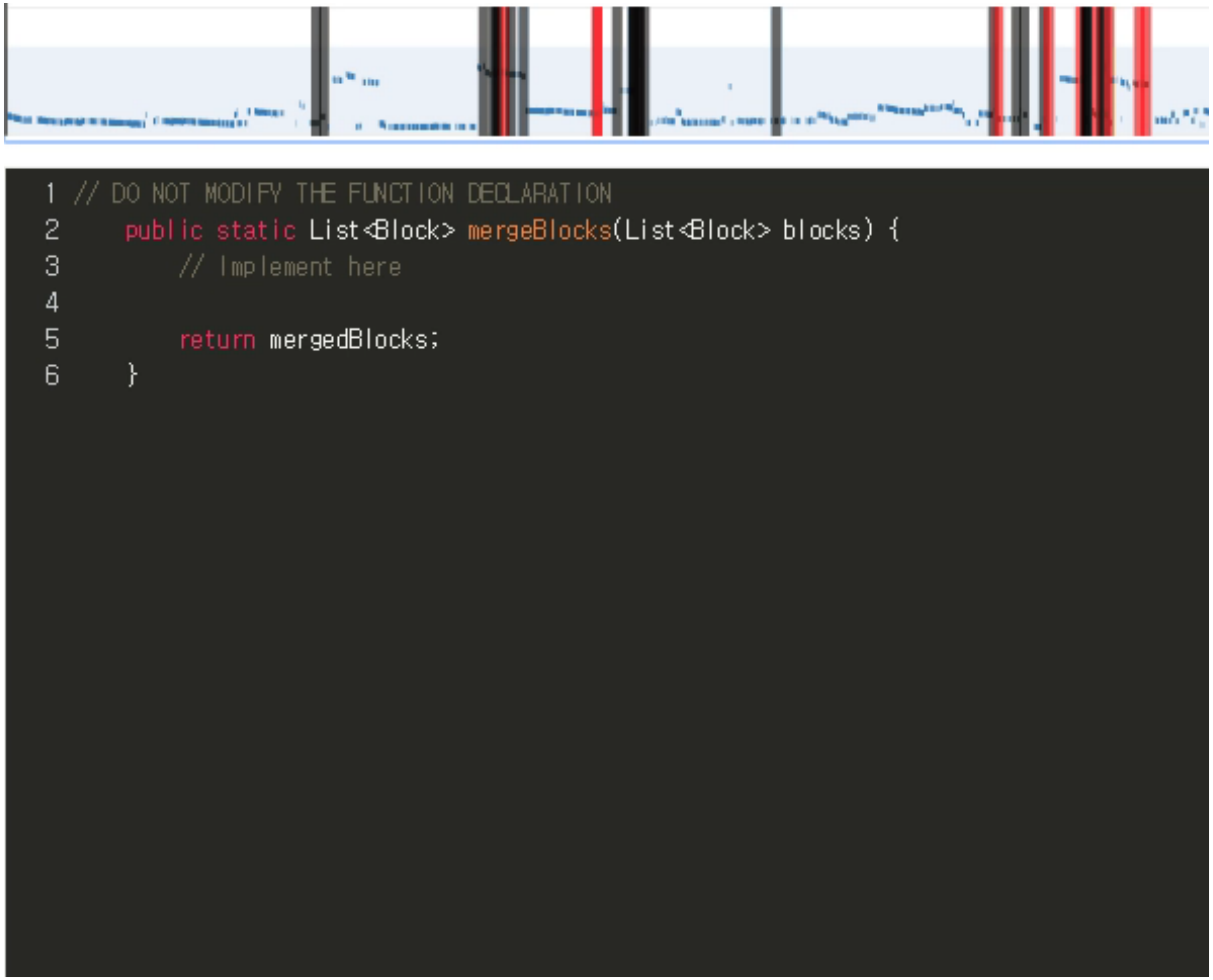
```
    public static List<Block> mergeBlocks(List<Block> blocks) {
        List<Block> mergedBlocks = new ArrayList<Block>();
        List<Block> sortedBlocks = new ArrayList<Block>();

        int inputLength = blocks.size();
        for (int i=0; i<inputlength; i++){
            Block min = blocks.get(0);
            int minInde
            for (Block b1 : blocks) {
                if (min.start > b1.start) {
                    min = b1;
                }
            }
            blocks.remove()
            sortedBlocks.add(min);
        }

        for (Block input : blocks) {
            for (Block temp : mergedBlocks) {
                if (input.start <= temp.start && input.end >= temp.end
                    temp.start = input.start;
                    temp.end = input.end;
                }
                else if (input.start <= temp.start && input.end <= tem
```

Execution events

Selection-based history tracking

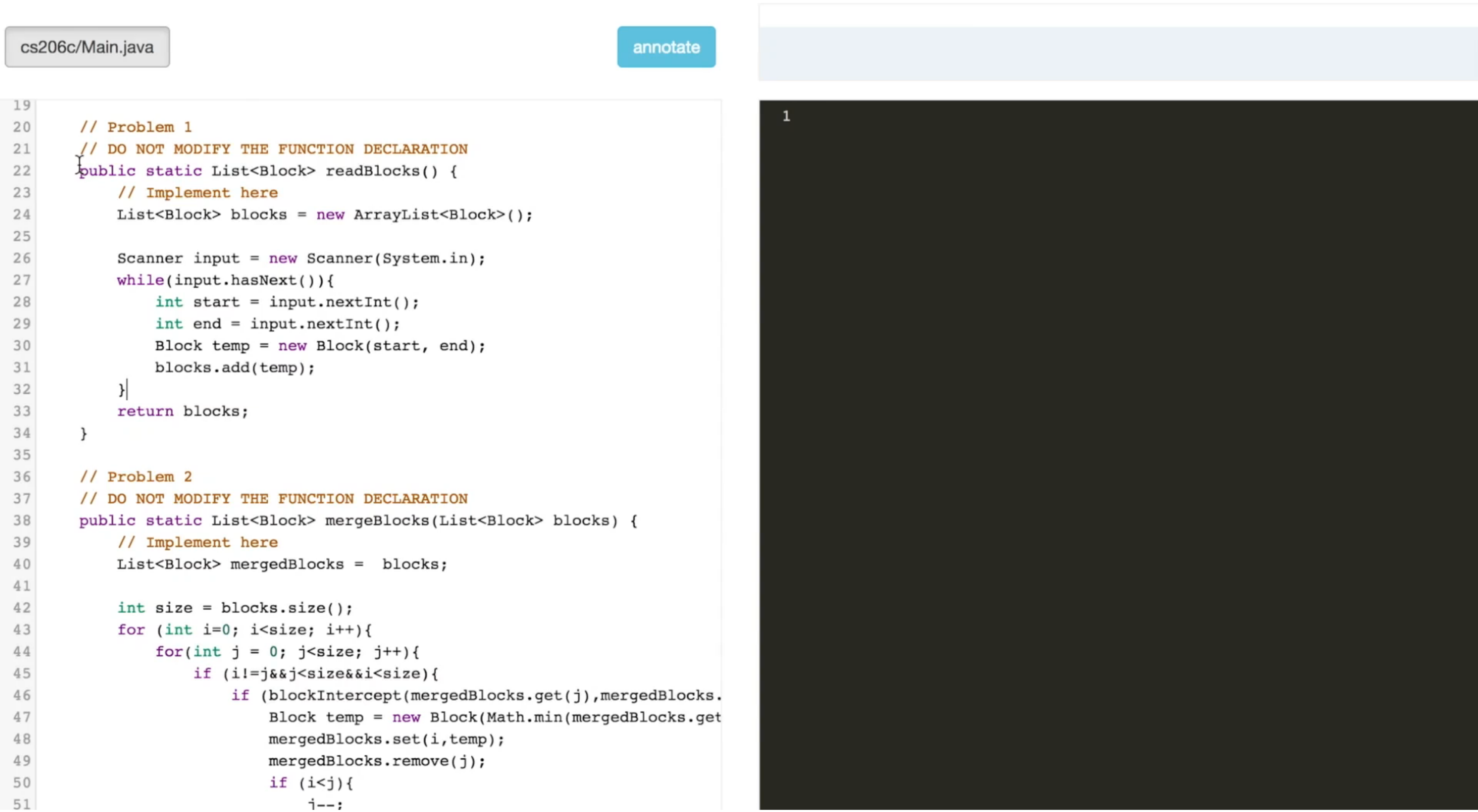
Character-level code history

Character-Level Code History



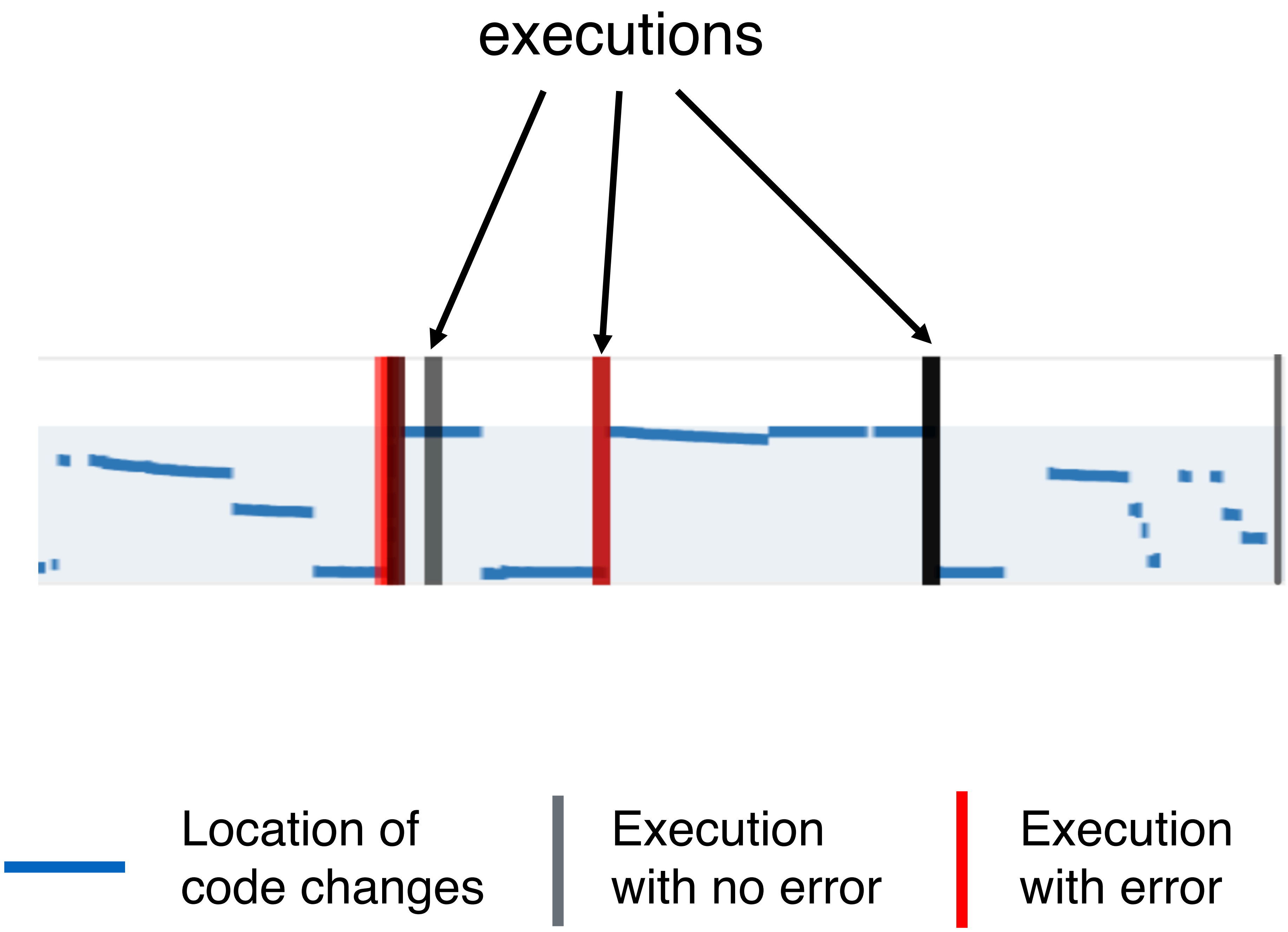
```
// DO NOT MODIFY THE FUNCTION DECLARATION
    public static List<Block> mergeBlocks(List<Block> blocks) {
        // Implement here

        return mergedBlocks;
    }
```

Selection-Based History Tracking



```
// Problem 1
// DO NOT MODIFY THE FUNCTION DECLARATION
public static List<Block> readBlocks() {
    // Implement here
    List<Block> blocks = new ArrayList<Block>();

    Scanner input = new Scanner(System.in);
    while(input.hasNext()){
        int start = input.nextInt();
        int end = input.nextInt();
        Block temp = new Block(start, end);
        blocks.add(temp);
    }
    return blocks;
}

// Problem 2
// DO NOT MODIFY THE FUNCTION DECLARATION
public static List<Block> mergeBlocks(List<Block> blocks) {
    // Implement here
    List<Block> mergedBlocks =  blocks;

    int size = blocks.size();
    for (int i=0; i<size; i++){
        for(int j = 0; j<size; j++){
            if (i!=j&&j<size&&i<size){
                if (blockIntercept(mergedBlocks.get(j),mergedBlocks.
                    Block temp = new Block(Math.min(mergedBlocks.get
                    mergedBlocks.set(i,temp);
                    mergedBlocks.remove(j);
                    if (i<j){
                        j--;
```

Execution Events



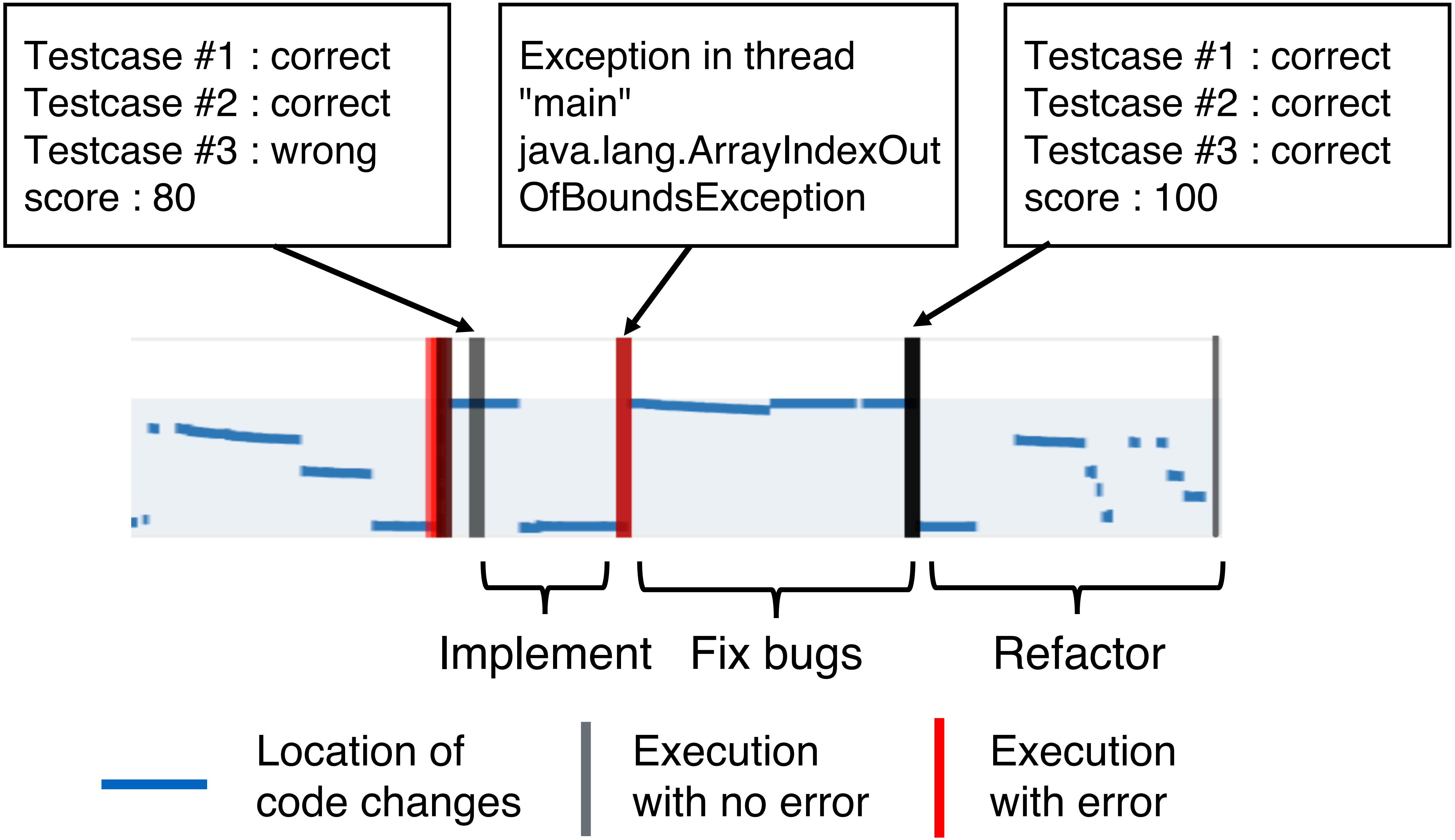
executions

Location of code changes

Execution with no error

Execution with error

Execution Events



Testcase #1 : correct
Testcase #2 : correct
Testcase #3 : wrong
score : 80

Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException

Testcase #1 : correct
Testcase #2 : correct
Testcase #3 : correct
score : 100

Location of code changes

Execution with no error

Execution with error

Evaluation

in a real classroom environment

Hypotheses

Visualization of code history

H1 - promotes **higher quality** of peer feedback

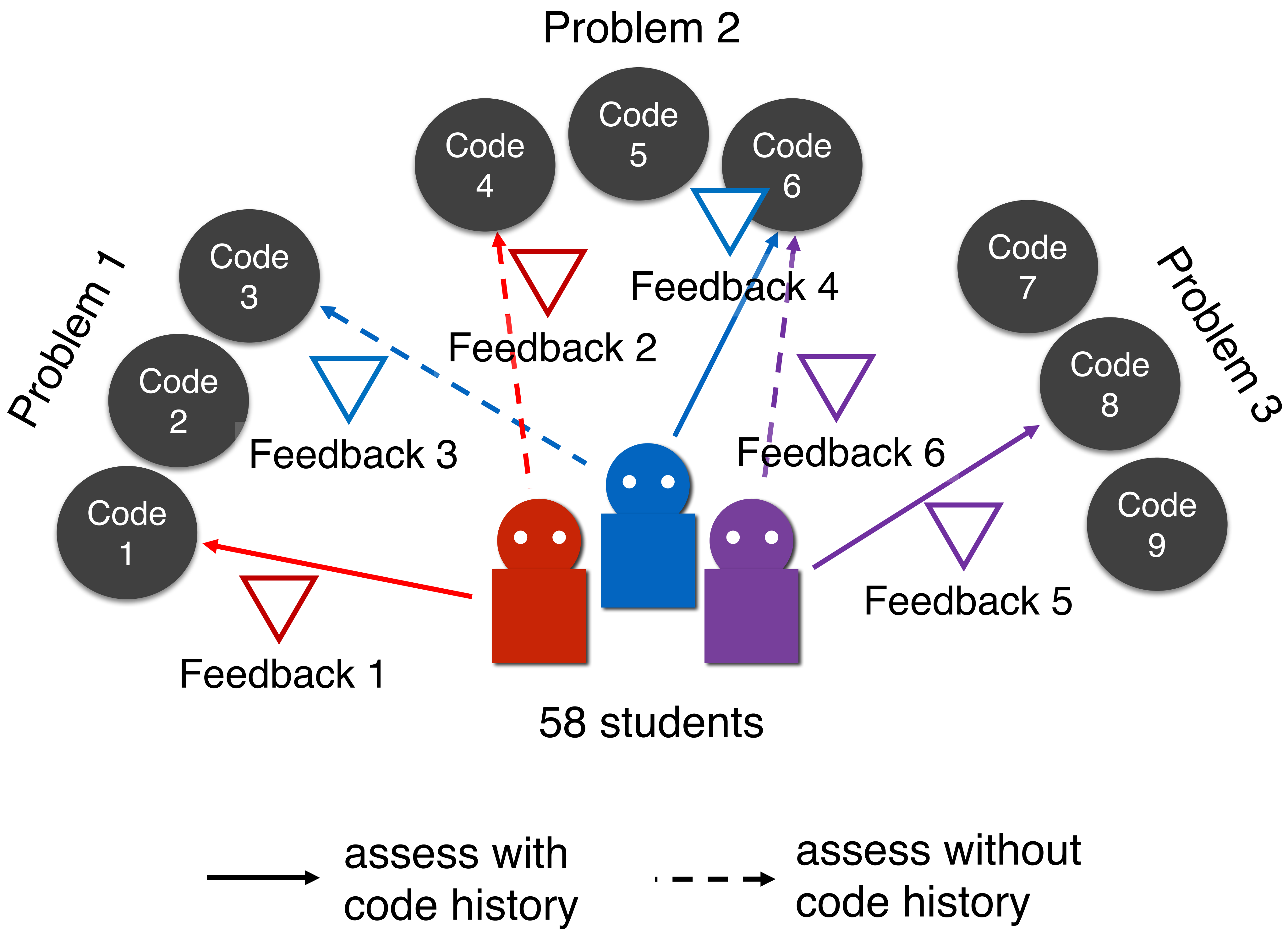
H2 - helps *student to get positive **learning** outcomes

H3 - improves the ****reliability** of peer assessment

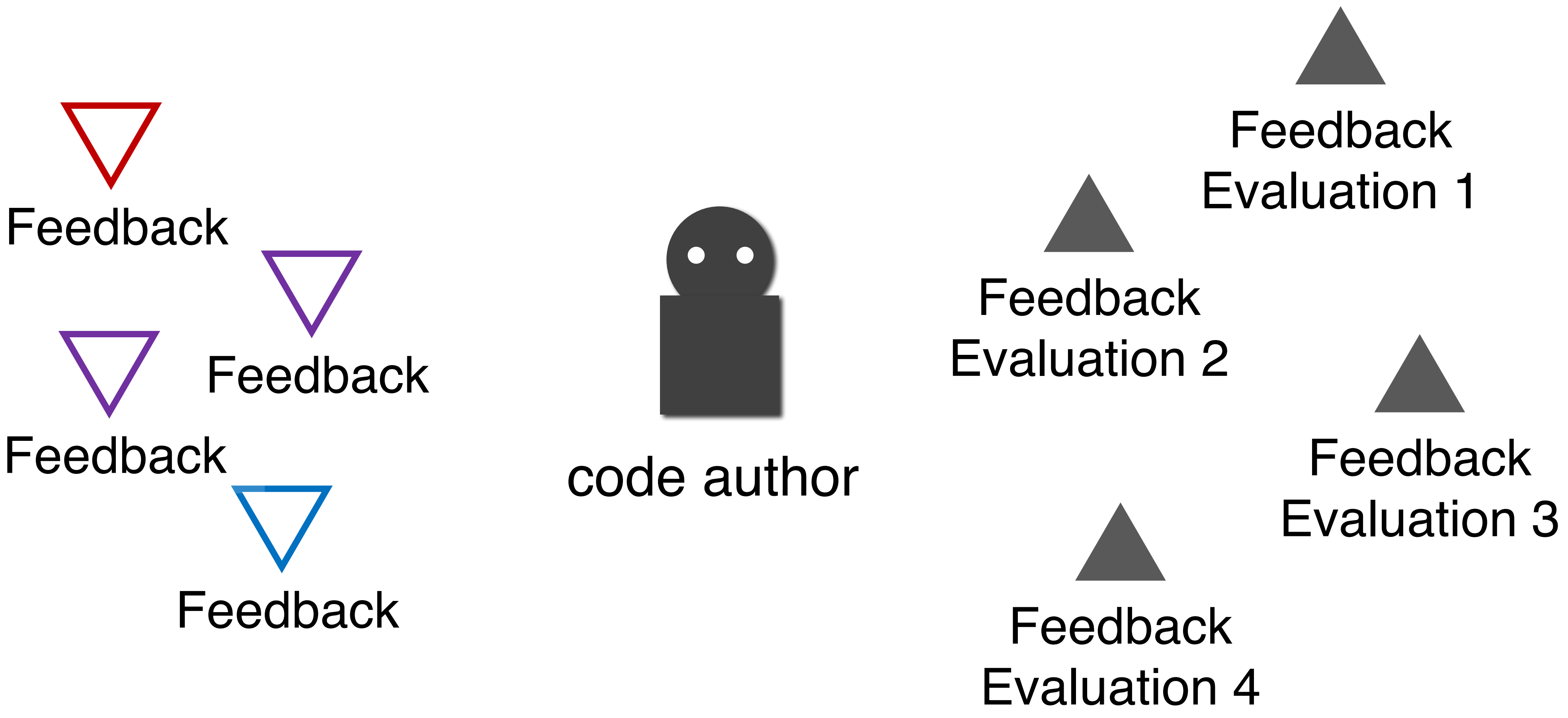
*student : assessor + code author

**reliability : the variance of scores given by peers

Step 1. Feedback Generation



Step 2. Feedback Evaluation



Feedback

Feedback

Feedback

Feedback

code author

Feedback Evaluation 1

Feedback Evaluation 2

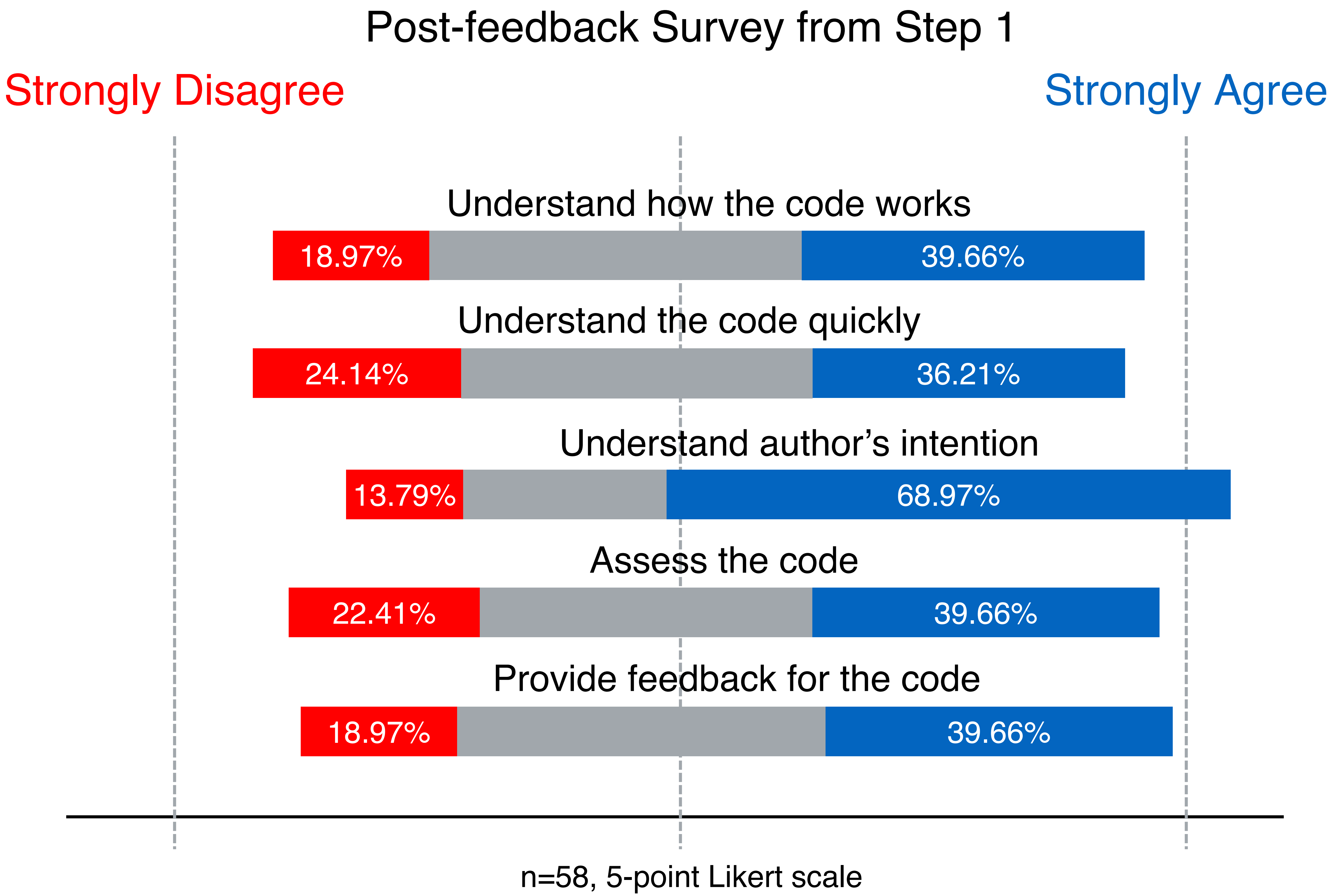
Feedback Evaluation 3

Feedback Evaluation 4

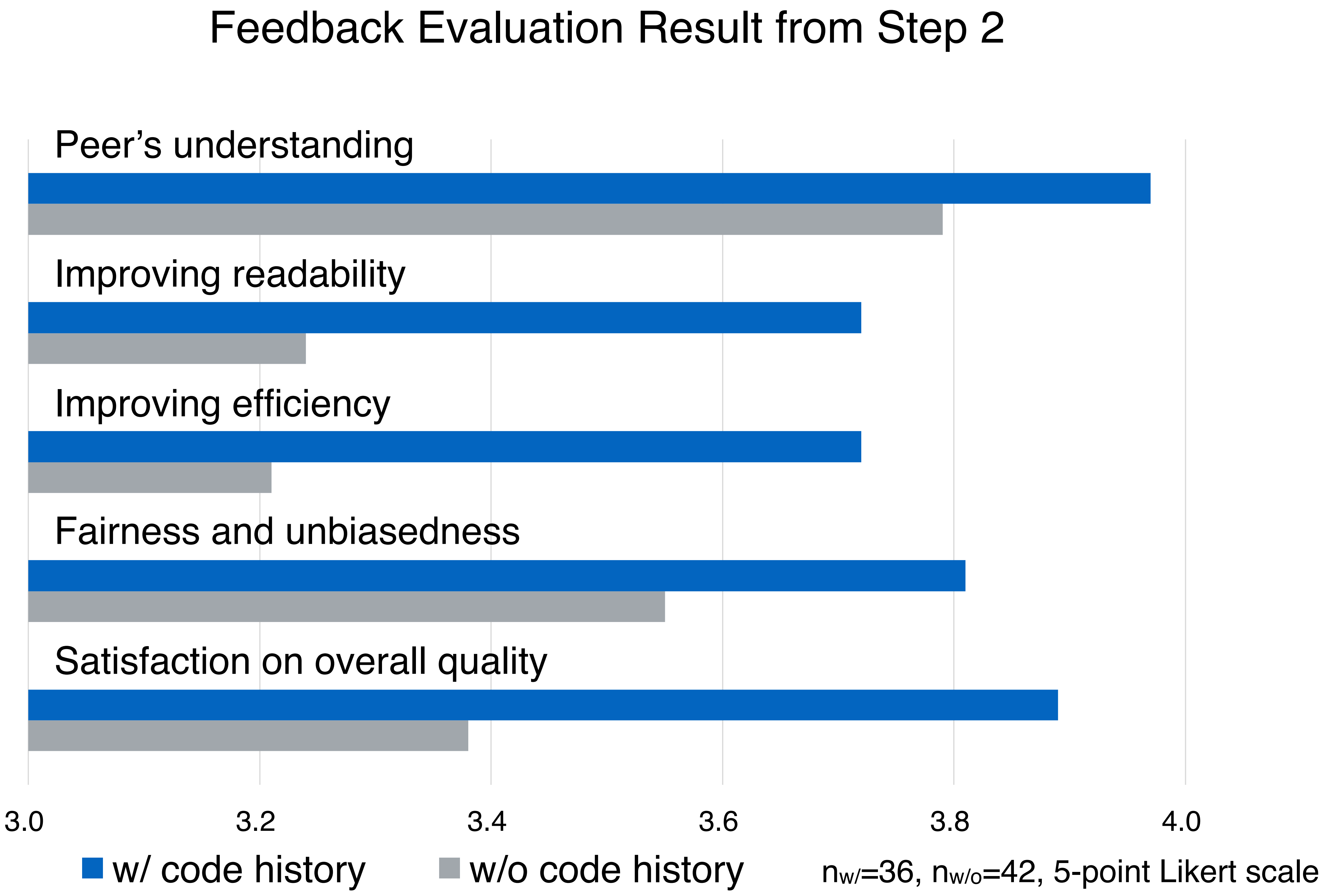
Analysis

hybrid method of quantitative and qualitative

H1: Eliph Promotes Higher Quality of Peer Feedback



H1: Eliph Promotes Higher Quality of Peer Feedback



$n_{w/}$=36, $n_{w/o}$=42, 5-point Likert scale

H1: Eliph Promotes Higher Quality of Peer Feedback

"How did browsing the code history help you assess?"

H1: Eliph Promotes Higher Quality of Peer Feedback

"How did browsing the code history help you assess?"

By inferring the intention of the code author

"It allowed me to understand ... why they implemented some of the functions." (Student 13)

H1: Eliph Promotes Higher Quality of Peer Feedback

"How did browsing the code history help you assess?"

By following the thought process of the code author

"… was helpful in understanding the author’s flow of thought" (Student 23)

H1: Eliph Promotes Higher Quality of Peer Feedback

"How did browsing the code history help you assess?"

By seeing the trial-and-error of the code author

"... I was able to understand where the author had been mistaken." (Student 4)

H1: Eliph Promotes Higher Quality of Peer Feedback

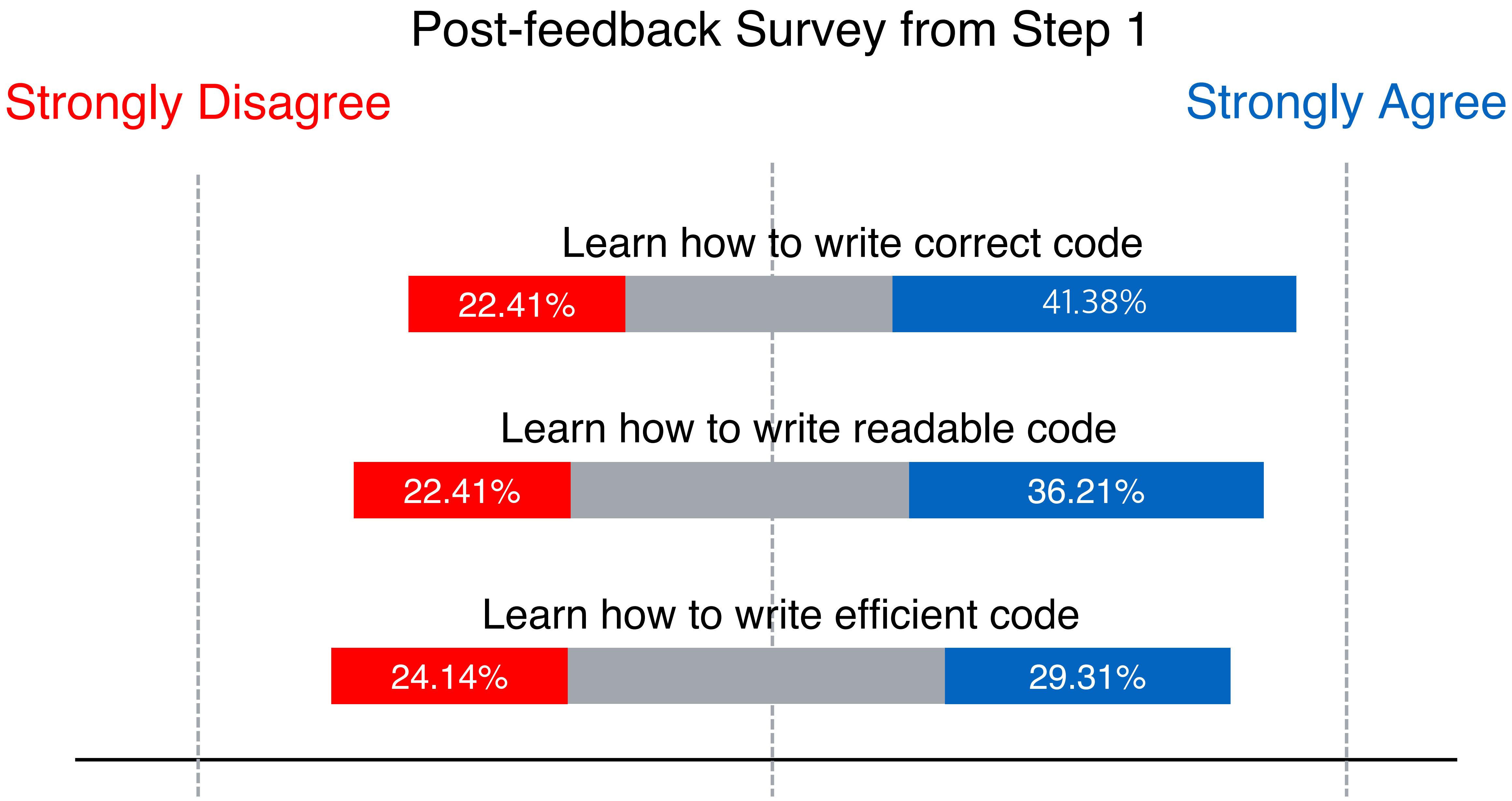
"How did browsing the code history help you assess?"

By understanding the code more easily

"In cases of code with poor readability, I had to browse its code history…" (Student 58)

"… I didn't have to understand the entire code at once." (Student 57)

H2: Eliph Helps Students Get Learning Outcome



n=58, 5-point Likert scale

H2: Eliph Helps Students Get Learning Outcome

"How did browsing the code history help you learn?"

H2: Eliph Helps Students Get Learning Outcome

"How did browsing the code history help you learn?"

By seeing how to write a readable code

"I learned some techniques such as naming variables, …, splitting code into small pieces, which could prevent potential problems as the code gets bigger" (Student 14)

H2: Eliph Helps Students Get Learning Outcome

"How did browsing the code history help you learn?"

By seeing similar ways of coding

"I realized that people write code using steps in different order. I learned more from code written by someone who codes more like myself." (Student 48)

H2: Eliph Helps Students Get Learning Outcome

"How did browsing the code history help you learn?"

By seeing how to overcome errors in specific situations

"... watching the trials and errors gave me insights into particular cases where some approaches simply don’t work." (Student 33)

H2: Eliph Helps Students Get Learning Outcome

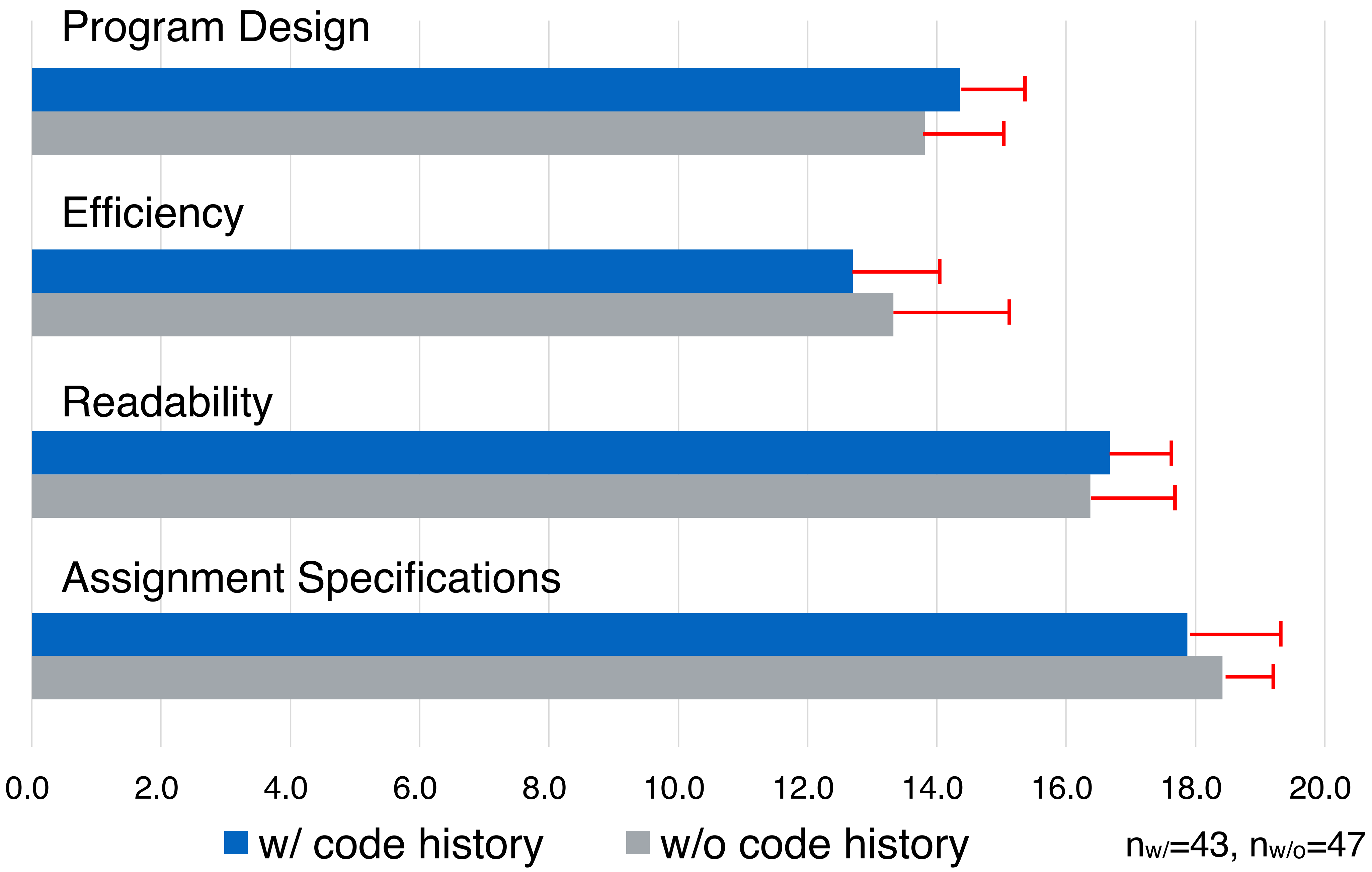
"If browsing the code history **did not** help you learn, why?"

"If a well-written code is given, I could see the process of writing good code by looking only at the final version of the code" (Student 51)

”... it contains wrong or inefficient code." (Student 44)

H3: Eliph Does Not Improve Reliability of Assessment

Code Assessment Result from Step 1



Conclusion

- We have introduced **Eliph**, a web-based peer assessment system with **code history visualization**.
- We have showed that Eliph has multiple benefits,
  - Looking at the code history helps student assessor **understand the code structure** as well as the **author's intention** more clearly.
  - **Overall quality of feedback** is higher when evaluated with the code history.
  - Evaluators feel that looking at the code history is helpful for their own learning.

Eliph: Effective Visualization of Code History for Peer Assessment in Programming Education

Jungkook Park (School of Computing, KAIST)

Yeong Hoon Park (School of Computing, KAIST)

Suin Kim (School of Computing, KAIST)

Alice Oh (School of Computing, KAIST)

pjknkda@kaist.ac.kr

Step 1. Feedback Generation

Code Assessment (Feedback) Criteria

A. Program Specification / Correctness (30pt) : Auto-graded

B. Program Design (20pt) + Comments
- Excellent (100%), Adequate (80%), Poor (60%), Not Met (0%)

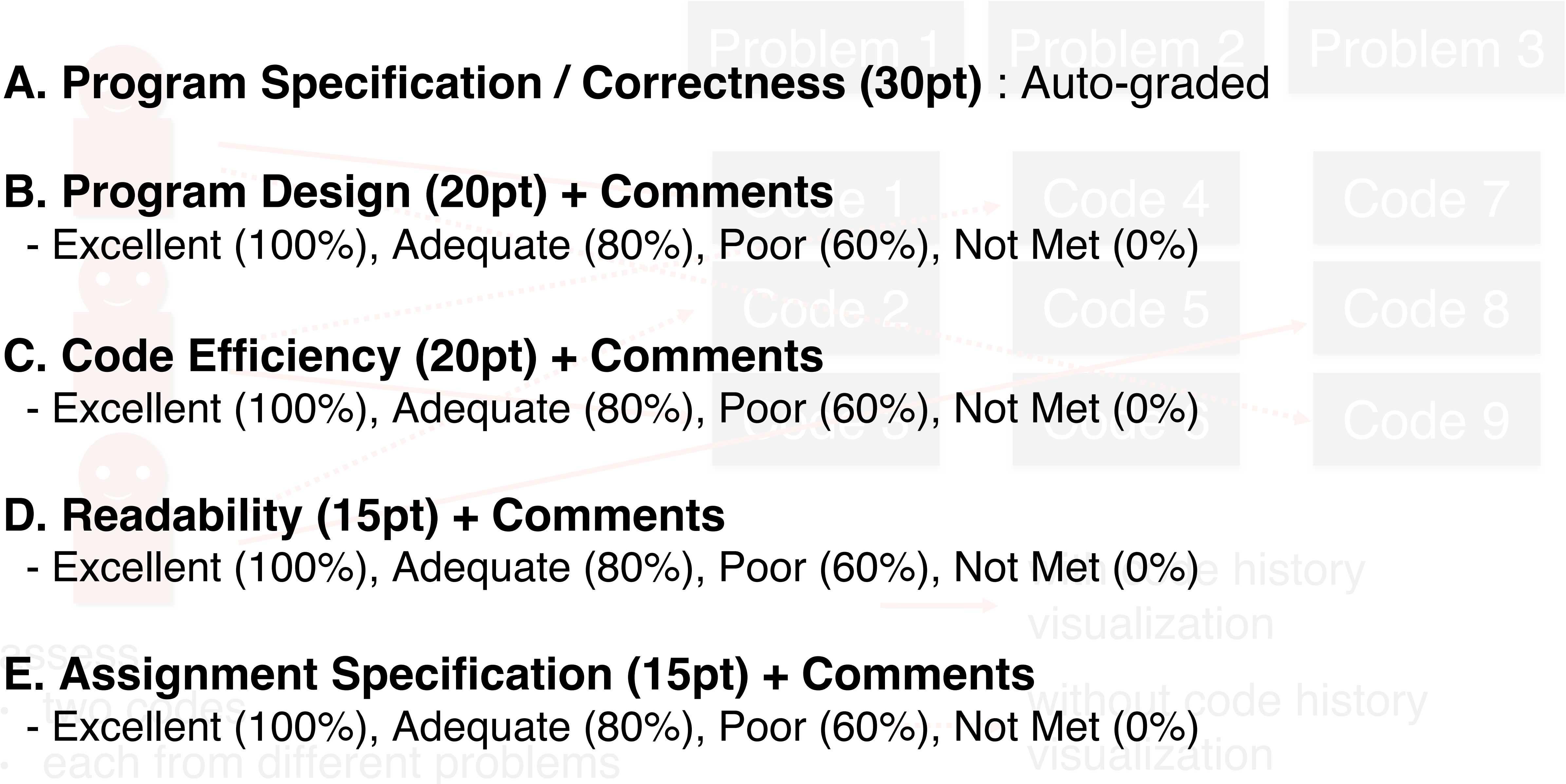
C. Code Efficiency (20pt) + Comments
- Excellent (100%), Adequate (80%), Poor (60%), Not Met (0%)

D. Readability (15pt) + Comments
- Excellent (100%), Adequate (80%), Poor (60%), Not Met (0%)

E. Assignment Specification (15pt) + Comments
- Excellent (100%), Adequate (80%), Poor (60%), Not Met (0%)

Undergraduate CS course
(Data Structure)

58 students



assess
- two codes
- each from different problems
- one with history / one without history

Step 1. Feedback Generation

Step 1: Post-feedback Survey

Section A. Peer Assessment

Q1 ~ Q5 (5-point Likert scale):
{ To understand how code works, To understand the code quckly, …} + *browsing the code history* was helpful than *viewing the last version of the code*.

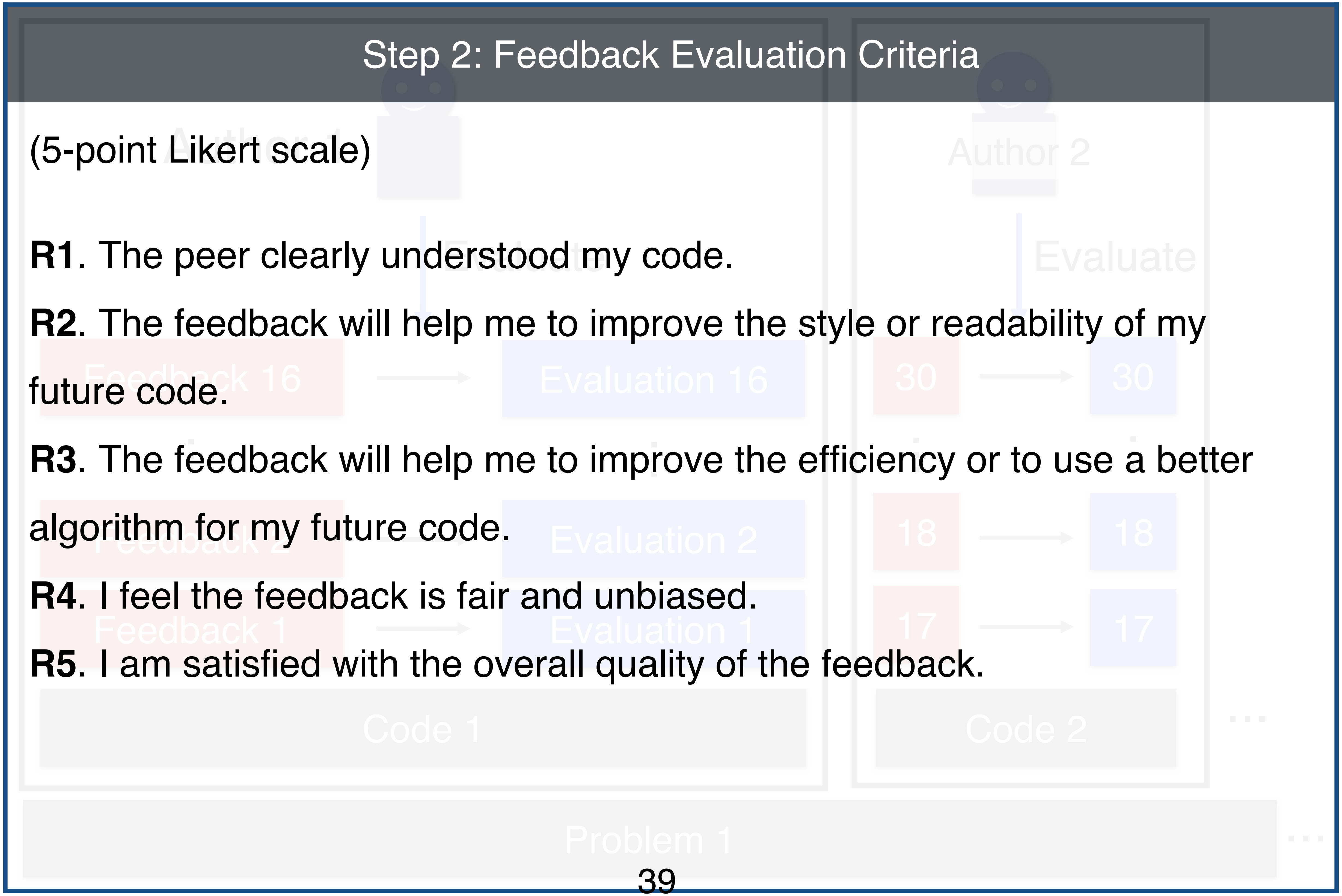
How did *browsing the code history* help you assess the code? If it did not, why?

Section B. Learning with Assessment

Q6 ~ Q8 (5-point Likert scale):
{ To learn how to write correct code, To learn how to write readable code, …} + *browsing the code history* was helpful than *viewing the last version of the code*.

How did *browsing the code history* help you learn to write a good the code? If it did not, why?

Step 2. Feedback Evaluation



Step 2: Feedback Evaluation Criteria

(5-point Likert scale)

R1. The peer clearly understood my code.

R2. The feedback will help me to improve the style or readability of my future code.

R3. The feedback will help me to improve the efficiency or to use a better algorithm for my future code.

R4. I feel the feedback is fair and unbiased.

R5. I am satisfied with the overall quality of the feedback.

Quantitative Findings : Post-Evaluation Survey

Browsing the code history was **more helpful than** viewing the last version of the code to

Question	Pos.(%)	Neg.(%)	Mean (SD)
understand author's intention of the code	68.97	13.79	3.86 (1.06)
learn how to write correct code	41.38	22.41	3.22 (0.89)
understand how the code works	39.66	18.97	3.19 (0.96)
provide feedback for the code	39.66	18.97	3.19 (0.97)
...	...	...	...
learn how to write efficient code	29.31	24.14	3.09 (0.92)

n=58, 5-point Likert scale

Quantitative Findings : Post-Evaluation Survey

support H1(Quality), H2(Learning)

Browsing the code history was more helpful than viewing the last version of the code to

More positive response than negative for all questions

Question	Pos.(%)	Neg.(%)	Mean (SD)
understand author's intention of the code	68.97	13.79	3.86 (1.06)
learn how to write correct code	41.38	22.41	3.22 (0.89)
understand how the code works	39.66	18.97	3.19 (0.96)
provide feedback for the code	39.66	18.97	3.19 (0.97)
...	...	...	...
learn how to write efficient code	29.31	24.14	3.09 (0.92)

n=58, 5-point Likert scale

Quantitative Findings : Feedback Evaluation

Criterion	Exp. Group	Control Group	P-value
Peer's understanding	3.97	3.79	0.33
Help to improving readability	3.72	3.24	**0.04**
Help to improving efficiency	3.72	3.21	**0.05**
Fairness and unbiasness	3.81	3.55	0.31
Satisfaction on overall quality	3.89	3.38	**0.04**

n_{exp}=36, $n_{control}$=42, 5-point Likert scale

Quantitative Findings : Feedback Evaluation

support H1(Quality)

Significant effect toward improving "style" and "efficiency" of the code

Criterion	Exp. Group	Control Group	P-value
Peer's understanding	3.97	3.79	0.33
Help to improving readability	3.72	3.24	**0.04**
Help to improving efficiency	3.72	3.21	**0.05**
Fairness and unbiasness	3.81	3.55	0.31
Satisfaction on overall quality	3.89	3.38	**0.04**

n_{exp}=36, $n_{control}$=42, 5-point Likert scale

Quantitative Findings : Feedback Evaluation

Criterion	Exp. Group	Control Group	P-value
Peer's understanding	3.97	3.79	0.33
Help to improving readability	3.72	3.24	**0.04**
Help to improving efficiency	3.72	3.21	**0.05**
Fairness and unbiasedness	3.81	3.55	0.31
Satisfaction on overall quality	3.89	3.38	**0.04**

support H1(Quality)

Significant effect toward the satisfaction on the quality of feedback

n_{exp}=36, $n_{control}$=42, 5-point Likert scale

Quantitative Findings : Assessment Statistics

Assessment Criterion	Avg. Score		P-value	
	Exp. Group	Control Group	T-Test	Levene-Test
Program Design	18.42	17.87	0.282	0.286
Efficiency	16.37	16.68	0.667	0.539
Readability	13.33	12.70	0.234	0.494
Assignment Specifications	13.81	14.36	0.309	0.298
Σ	61.93	61.62	0.846	0.710

$n_{exp}=43$, $n_{control}=47$

Quantitative Findings : Assessment Statistics

reject H3(Reliability)

No significant difference in both mean and variance

Assessment Criterion	Avg. Score		P-Value	
	Exp. Group	Control Group	T-Test	Levene-Test
Program Design	18.42	17.87	0.282	0.286
Efficiency	16.37	16.68	0.667	0.539
Readability	13.33	12.70	0.234	0.494
Assignment Specifications	13.81	14.36	0.309	0.298
Σ	61.93	61.62	0.846	0.710

$n_{exp}=43$, $n_{control}=47$

Qualitative Findings : Post-Evaluation Survey

"How did **browsing the code history** help you **assess**?"

Intention	*It allowed me to understand ... why he implemented some of the functions.*
Though process	*it was helpful in understanding the author's flow of thought.*
Trial-and-error	*... I was able to understand where the author was mistaken.*
Code readability	*... I didn't have to understand the entire code at once, ...*

Qualitative Findings : Post-Evaluation Survey

"If it did not (help you **assess**), why?"

Since it wasn't a big project, I couldn't get much extra information out of the code history.

It did not help too much because the code was easy to understand.

... I think code history is something that should be hidden. ...

Qualitative Findings : Post-Evaluation Survey

"How did **browsing the code history** help you **learn**?"

Writing readable code	*I learned some techniques such as naming variables, splitting code into small pieces, ...*
Different code styles	*... I feel like I came to realize the right way how one should write code.*
Trial-and-error	*... watching the trials and errors gave me insight into particular cases some approach doesn't work.*

Qualitative Findings : Post-Evaluation Survey

"If it did not (help you **learn**), why?"

Not much to learn

If a well-written code is given, I could know the process and how to write code only seeing the final version of the code.

Poorly written code

Unless peer's code is perfect, seeing that code history does not seem to have learned something.

Quality of Peer Feedback: Quantitative Analysis

H1. Does Eliph promote higher quality of peer feedback?

YES!

Step 1: Post-feedback Survey

	Pos.(%)	Neg.(%)
Q1	**39.66**	18.97
Q2	**36.21**	24.14
Q3	**68.97**	13.79
Q4	**39.66**	22.41
Q5	**39.66**	18.97

n=58, 5-point Likert scale

Step 2: Feedback Evaluation Result

Criterion	w/ code history	w/o code history
Peer's understanding	3.97	3.79
Help to improving readability	***3.72**	3.24
Help to improving efficiency	†**3.72**	3.21
Fairness and unbiasness	3.81	3.55
Satisfaction on overall quality	***3.89**	3.38

$n_{w/}$=36, $n_{w/o}$=42, 5-point Likert scale

*Q1 - Q5 : To do …, w/ code history was helpful than w/o code history.

Learning Outcome: Quantitative Analysis

H2. Does Eliph help student to get positive learning outcomes?

YES!

Step 1: Post-feedback Survey

	Pos.(%)	Neg.(%)
Q6	**41.38**	22.41
Q7	**36.21**	22.41
Q8	**29.31**	24.14

n=58, 5-point Likert scale

*Q6 - Q8 : To learn how to …, w/ code history was helpful than w/o code history.

Reliability of Peer Assessment: Quantitative Analysis

H3. Does Eliph improve the reliability of peer assessment?

No.

No significant difference in the variance

Step 1: Code Assessment Result

Assessment Criterion	Avg. Score (SD)		P-value (Levene-Test)
	w/ code history	w/o code history	
Program Design (20pt)	18.42 (2.14)	17.87 (2.59)	0.286
Efficiency (20pt)	16.37 (2.97)	16.68 (3.72)	0.539
Readability (15pt)	13.33 (2.08)	12.70 (2.78)	0.494
Assignment Specifications (15pt)	13.81 (2.97)	14.36 (1.85)	0.298
Σ	61.93 (7.45)	61.62 (7.64)	0.710

$n_{w/}=43$, $n_{w/o}=47$