# Automated measures of student elegance

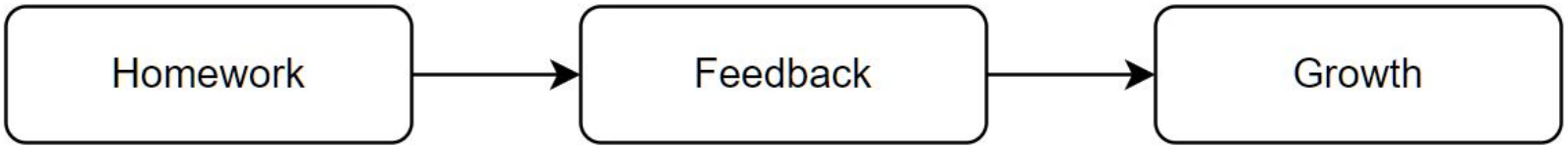
Nat Hurtig and Joe Hollingworth

# Before we start…

- This is an ongoing project
- We’d love to hear your feedback and ideas on our work so far

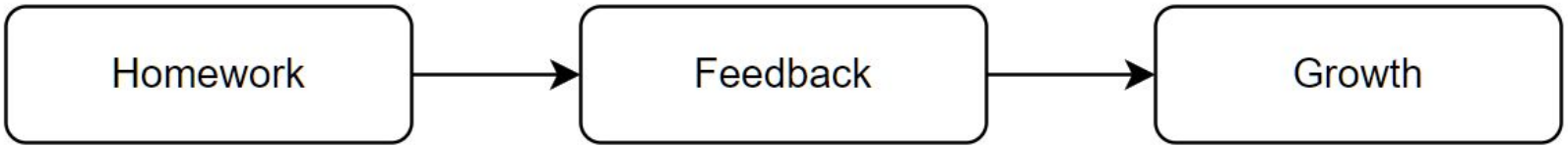
# Background

- Students need feedback on their formative assignments
- We'd like to give feedback by hand



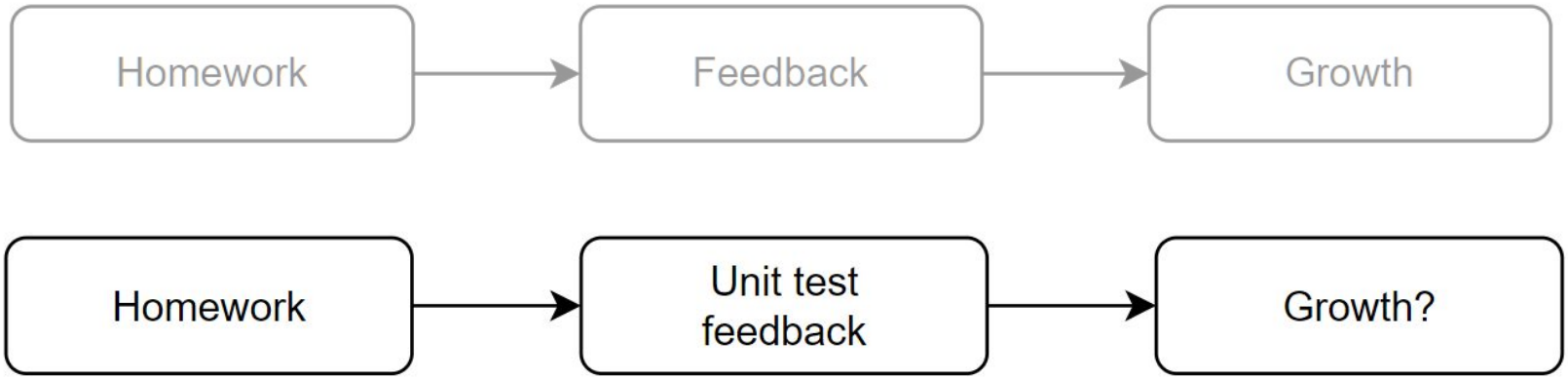
# The problem

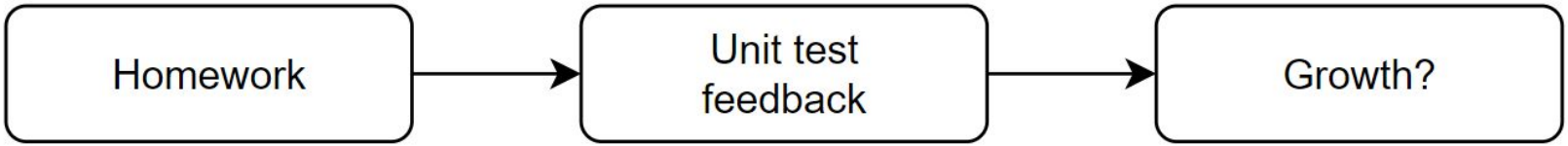
- Computer science educators aren’t grading essays
- Lots of students and many assignments makes grading by hand infeasible

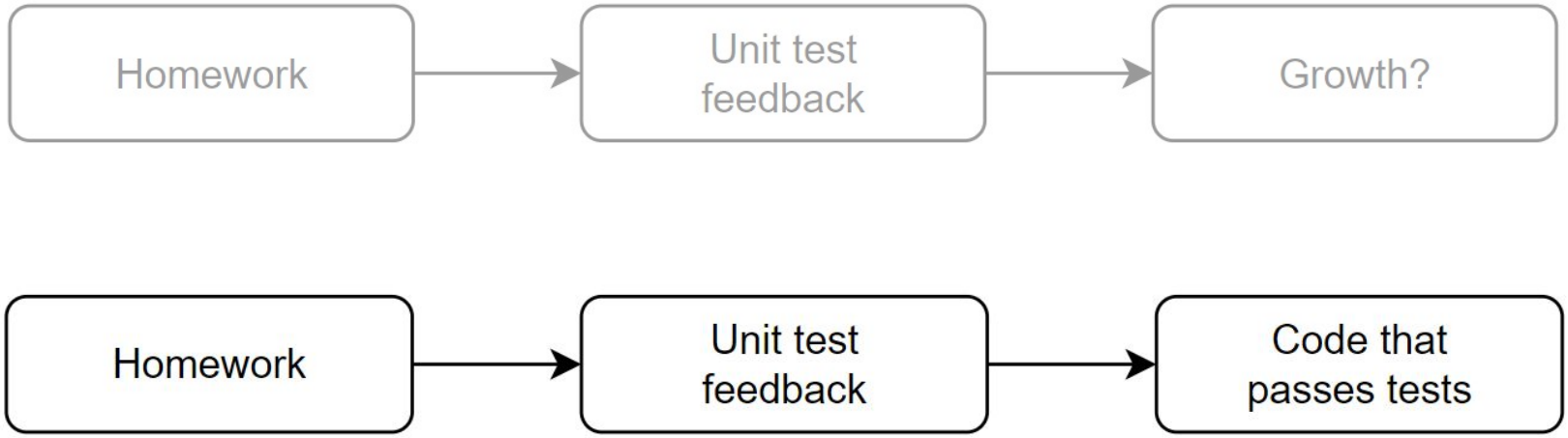


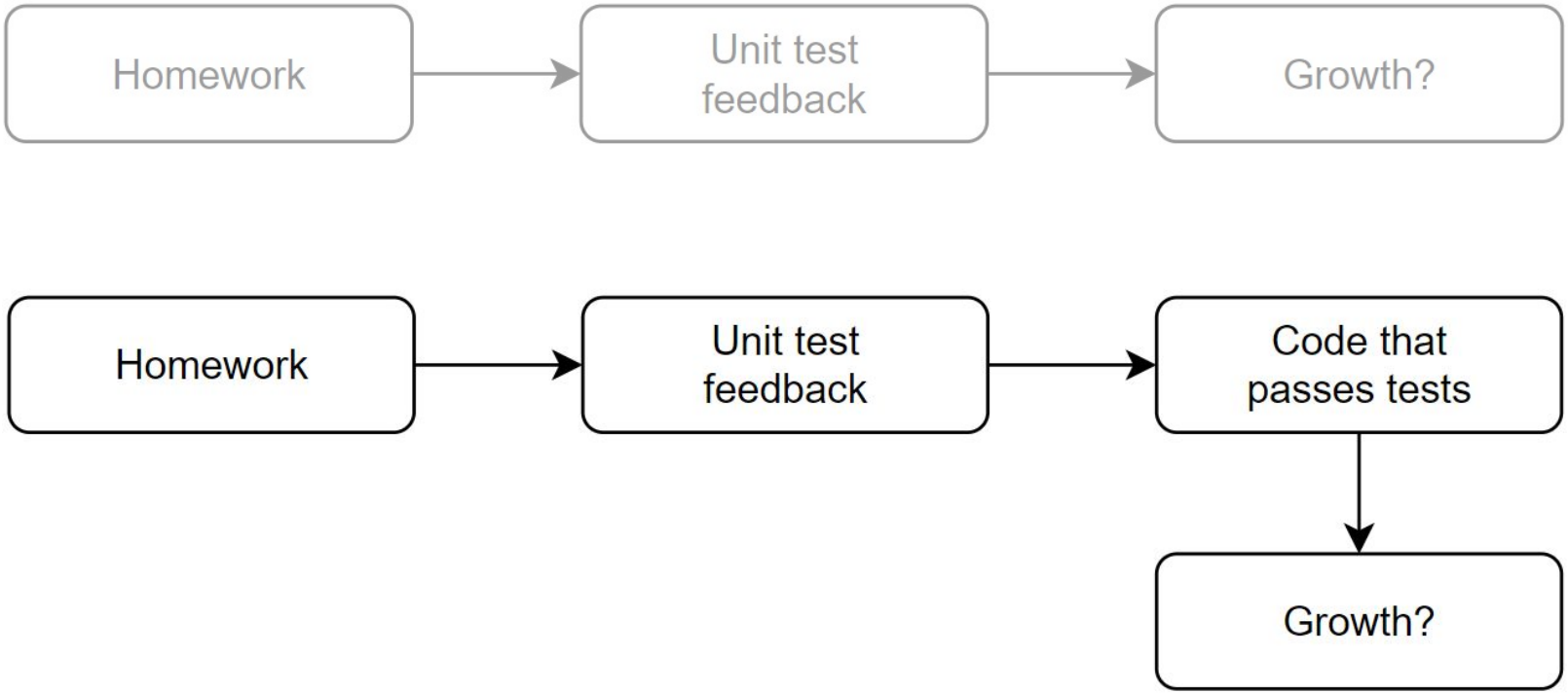
# Enter unit tests

- We use automation to give students instant feedback
- Students see whether their code passes unit tests on demand




Homework
Unit test feedback
Growth?


Homework
Unit test feedback
Growth?
Homework
Unit test feedback
Code that passes tests


Homework
Unit test feedback
Growth?
Homework
Unit test feedback
Code that passes tests
Growth?

# More than correctness

- Passing unit tests is important, but professional software developers aim to write code that is:
  - Maintainable
  - Efficient
  - Readable
  - Secure
- Can basic unit tests encourage this?

# An example: mostCommonCharacter

- An exercise to teach Map objects for our second course in computer science

- Given a string, find the most common character

- Our expected solution: iteration over the string, updating a map

    - Example: on input “banana”, the built map would be {(‘n’,2),(‘b’,1),(‘a’,3)}.

# An example: mostCommonCharacter

- A solution from a real student

```
public static char mostCommonCharacter38(String input) {
    int[] alpha = new int[26];
    for(int i = 0; i < input.length(); i++) {
        if(input.charAt(i) == 'a') alpha[0]++;
        if(input.charAt(i) == 'b') alpha[1]++;
        if(input.charAt(i) == 'c') alpha[2]++;
        if(input.charAt(i) == 'd') alpha[3]++;
        if(input.charAt(i) == 'e') alpha[4]++;
        if(input.charAt(i) == 'f') alpha[5]++;
        if(input.charAt(i) == 'g') alpha[6]++;
        if(input.charAt(i) == 'h') alpha[7]++;
        if(input.charAt(i) == 'i') alpha[8]++;
        if(input.charAt(i) == 'j') alpha[9]++;
        if(input.charAt(i) == 'k') alpha[10]++;
        if(input.charAt(i) == 'l') alpha[11]++;
        if(input.charAt(i) == 'm') alpha[12]++;
        if(input.charAt(i) == 'n') alpha[13]++;
        if(input.charAt(i) == 'o') alpha[14]++;
        if(input.charAt(i) == 'p') alpha[15]++;
        if(input.charAt(i) == 'w') alpha[22]++;
        if(input.charAt(i) == 'x') alpha[23]++;
        if(input.charAt(i) == 'y') alpha[24]++;
        if(input.charAt(i) == 'z') alpha[25]++;
    }
    int largest = 0;
    int numberAlpha = 0;
    for(int i = 0; i < alpha.length; i++) {
        if(alpha[i] > largest) {
            largest = alpha[i];
            numberAlpha = i;
        }
    }
```

# Our metric: Elegance

- There’s no standard definition of “elegance” in coding
- We do agree that unnecessary code is inelegant
  - Extra for loop
  - Useless variable
  - Redundant if statement

# Why do we care about inelegance?

- Encouraging student to limit their “if” and “for” statements pushes them to learn about the language they’re working in, or to reuse code they’ve already written
  - Example: Java’s getOrDefault for Maps
- Students might reflect on the code they write instead of chasing after correctness
  - Discourages guess-and-check strategies while coding
- Prepares students for environments where code needs to be more than functional

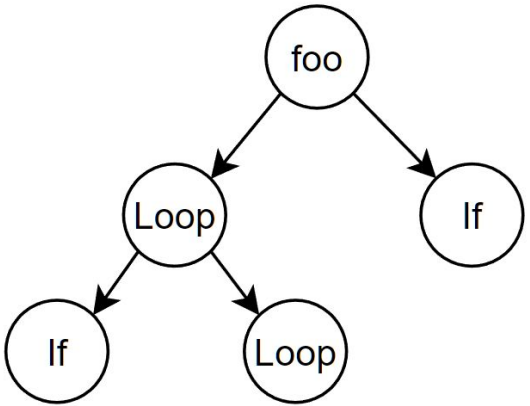
# How can we measure inelegance?

- Linting tools: mostly meant for professional software developers
- AI: not trustworthy or explainable

# Measuring inelegance

```
private int foo(int arg) {
    for (...) {
        if (...) {
            ...
        }
        while (...) {
            ...
        }
    }
    if (...) {
        ...
    } else {
        ...
    }
}
```

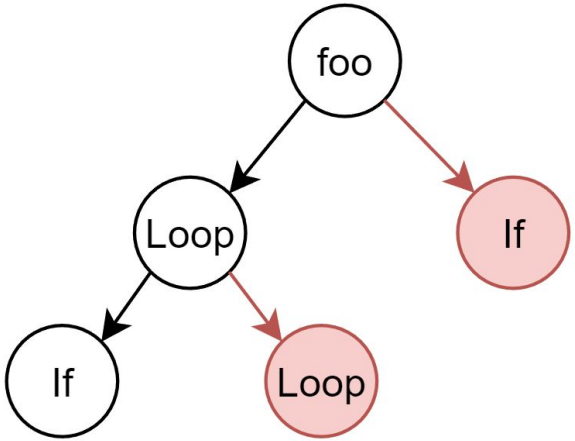
- We use a parser to generate an Abstract Syntax Tree (AST) for a student's code
- We use that to generate a tree of our own for each function
  - We record loops (for, while) and selection statements (if/else, ternary operator)



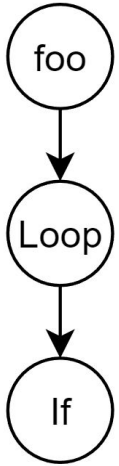
# Measuring inelegance

- The instructor writes an intended solution to each method
- We compare the student's tree against the instructor's intended solution tree, and count the extra statements

Student solution




Intended solution



# Examples

Unnecessary base case

```java
public static int firstDivisibleBy7752(int[] numbers) {
    if (numbers.length == 0) {
        return -1;
    }

    for (int i = 0; i < numbers.length; i++) {
        if (numbers[i] % 77 ==0) {
            return numbers[i];
        }
    }
    return -1;
}
```

Could have used substring

```java
public static ArrayList<String> threeCharacterStrings60(String input) {
    ArrayList<String> list = new ArrayList<>();
    if(input.length()  == 3) {
        list.add(input);
        return list;
    }else if(input.length()/3 < 1) {
        return list;

    }else {
        for(int i =0; i < input.length() - 2 ; i++) {
            String part = "";
            char a = input.charAt(i);
            char b = input.charAt(i+1);
            char c = input.charAt(i+2);
            String end = part+ a + b +c;
            list.add(end);
        }
    }
    return list;
}
```

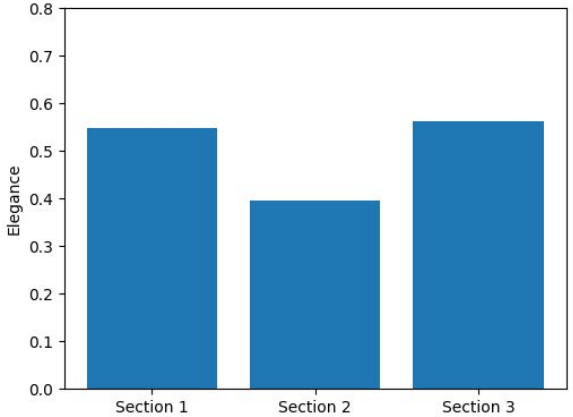
# Experiment

- We had 3 sections of students taking our object-oriented Java course
- On their Maps homework assignment, we issued section 2 a challenge; sections 1 and 3 were not treated
- We challenged section 2 to implement mostCommonCharacter with at most 12 lines of code, 1 for loop, and 1 if statement
- This required keeping a running maximum (as opposed to first building a map, and then iterating through the map) and using getOrDefault

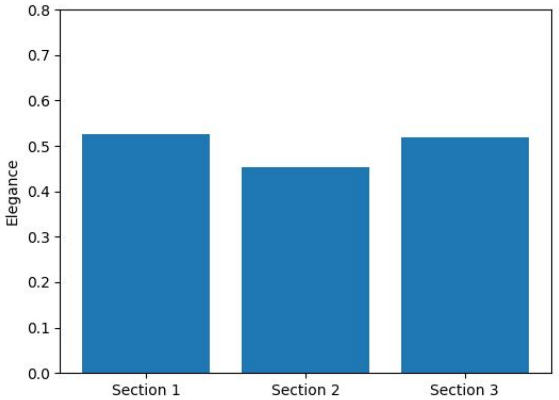
# Results

- On the methods they weren't challenged on, section 2 was less elegant
- Section 2 improved on the challenged method, but the results weren't spectacular

Other four methods




Challenged method



# Next Steps

Challenge Experiment

- We want to repeat the challenge experiment again
- The first time, the challenge was administered in class, with nothing in the starter file provided to the students
- During a repeat we would like to include comments in the starter code file that reinforces the challenge and provides the details of X lines of code,Y ‘for’/’while’ loops, and Z ‘if’ statements

Tool Support

- Create an IDE plugin that analyzes an operation for elegance
- This plugin would provide elegance feedback to the student during the development phase
- This evaluation would be based on a comparison of the students code versus the instructor’s intended solution