Homework 10 - Problem Description & Literature Review

Andrew Carter and Beryl Egerter

April 10, 2013

1 Problem Descriptions

1.1 Q1

1.1.1 Summary

Interview Question 1 (Q1) asks the interviewee to evaluate a short program. The program passes a function as an argument, a concept they might not be familiar with. The program is 18 lines long and takes up less than a page when printed.

1.1.2 Copy of Problem

The problem can be found in the Appendix attached below as Interview Question 1.

1.1.3 Solution Description

One (popular) way to determine the solution is to realize the only piece of code that get executed at the top level is the call to main on 18. Then following the execution of the program, main prints out the result of `func3([1,2,3,4])`. The functions func3 and func2 just add arguments, so that call is equivalent to calling `func1([1,2,3,4],4,func4)`. Note that func4 is being passed, thus when "f" is called in func1 it is really calling the function func4. Continuing the trace in func1, acc starts at 0 then adds `func4(i,4)` for each "i" in the list. Since func4 is just multiplication, this results in summing 4 times the elements of the list, which comes out to 40. When we unwind we just hit return statements until main, when we print 40.

1.2 Q4

1.2.1 Summary

Interview Question 4 (Q4) asks the interviewee to locate and fix a bug in a program that they are told plays connect four. The bug is that when pieces are dropped into the board, the program lets pieces be dropped in even if the column is full. The program is 70 lines long and covers two pages when printed.

1.2.2 Copy of Problem

The problem can be found in the Appendix attached below as Interview Question 4.

1.2.3 Solution Description

To find the correct solution we start at line 49 where the code starts executing. Some variables are set up for use within the Read-Eval-Print loop that follows Here the Board is initialized. It calls the __init__ function (line 3) which create a list of lists stored as the self.board variable. The initial lists are empty. It also stores the height and width of the board in class variables.

At line 55 we see that the program asks for input, the read part of the loop. Then, it calls board.drop(player,c). In the drop function (line 8), we can see that if the column selected is part of the board, the piece is appended to that column in the list of lists. This is analogous to dropping a piece into the board, meaning this is where the bug needs to be fixed by.

Since we check that the column is valid, we can also check that the column is not full by changing the drop function to include this line between lines 9 and 10: if len(self.board[column]) < self.height:, to make sure that pieces will not be dropped.

Other possible solutions that cause the drop function to not be called if the column is full would also be acceptable.

2 Literature Review

A number of research studies and papers have talked about code comprehension and debugging. The following papers cover both reviews of debugging and code evaluation and as such are highly connected to our work.

Mosemann and Wiedenbeck investigate code comprehension by novices with respect to navigation method (Mosemann & Wiedenbeck, 2001). They gave participants two different problems and a asked a number of questions. The students were assigned a navigation method, and evaluated based on the navigation method. They concluded that novices prefer sequential, and control flow, even though they have not been trained in the latter. This pertains to our study because it is a qualitative review of navigation methods on a single problem. Furthermore they combined the results of the problems because there was no interaction between program and navigation method, however we suggest that with more diverse methods there may be some interaction, but they only posed two different problems.

Fitzgerald, et al. discuss debugging methods among novices (Fitzgerald et al., 2008). They note an increased reliance on online resources for debugging, something that we did not provide. Their results show that bugs are relatively easy to fix, and that determining how programmers discover where a bug is located is more useful. Some student strategies exhibited that we could not possibly replicate in our study are using println statements and tracing using a debugger. They do confirm that students often used a tracing strategy on a variety of different debugging problems.

Rajlich and Wilde talk about concepts both as code formations and domain knowledge (Rajlich & Wilde, 2002). They argue that experts do not understand the entirety of the code they are debugging but simply map their domain knowledge concept on to code concepts to narrow down the code areas they must look at. This mapping of domain knowledge to code occurred in interview ACBE3 as the student connected functions within the code to actions taken in a connect four game. These mappings allowed the student to focus only on a single function instead of understanding the entire program.

Mayrhauser and Vans notice that often program comprehension involves multiple ways of thinking about the code (von Mayrhauser & Vans, 1995). In addition to explaining an umber of them and how they believe they work together, they perform a study and provide results of how often each of the strategies was used by a programmer looking for a section of code within a 40000 line program in a two hour period of time. While most of our interviews were far too short for our subjects to switch between different strategies, we did occasionally see them, and this paper provides information about what might occur if our problems and interviews had been longer.

These papers are relevant to our work because they cover various studies about code debugging, evaluation, and the strategies used in either case. Our work builds on these papers by comparing debugging and evaluation strategies directly instead of treating them as separate topics.

3 Appendix (for reference)

3.1 Interview Question 1

Code given to participants did not include line numbers.
Code given to ACBE1, ACBE2, and ACBE3 was in color. Code given to ACBE4 was in grayscale.
Verbal prompt was given before handing code to participant:
"For this question we would like you to familiarize yourself with some Python code. Please explain to us what you think this code does."

```
0   def func2(list , num):
1     return func1(list , num, func4)
2
3   def func4(a, b):
4     return a * b
5
6   def func1(list , num, f):
7     acc = 0
8     for i in list:
9         acc += f(i, num)
10    return acc
11
12  def main():
13    print(func3([1,2,3,4]))
14
15  def func3(list):
16    return func2(list , 4)
17
18  main()
```

3.2 Interview Question 4

Code given to participant did not contain line numbers.
Code given to ACBE1, ACBE2, and ACBE3 was in color. Code given to ACBE4 was in grayscale. Code given to ACBE1 and ACBE2 had double equals signs that were joined together. Code given to ACBE3 and ACBE4 had spaces between the equals signs.
Verbal prompt was given before handing code to participant:
"For this question we would like to have you look at some code in Python. This is the scenario: You acquired a connect 4 program from a friend. However, the friend has warned you that you can put too many pieces in a column. Determine a possible fix for this bug so that you can enjoy your connect 4 program."

```
0   #!/bin/env python3
1
2   class Board(object):
3     def __init__(self, width=7, height=6):
4       self.board = [[] for i in range(width)]
5       self.width = 7
6       self.height= 6
7
8     def drop(self, player, column):
9       if column < len(self.board):
10        self.board[column].append(player)
11        return True
12      return False
13
14    def __str__(self):
15      result = ""
16      for r in reversed(range(self.height)):
17        result += "|"
18        for c in range(self.width):
19          if r < len(self.board[c]):
20            result += self.board[c][r]
21          else:
22            result += "␣"
23          result += "|"
24        result += "\n"
25      result += "-" * (2 * self.width + 1)
26      return result
27
28    def full(self):
29      return all(len(col) >= self.height for col in self.board)
30
31    def score(self, player):
32      for c in range(self.height):
33        for r in range(len(self.board[c])):
34          p = self.board[c][r]
35          for dc,dr in ((0,1),(1,0),(1,1),(1,-1)):
36            for i in range(1,4):
37              nc = c + i*dc
38              nr = c + i*dr
39              if nc < 0 or self.width <= nc:
40                break
41              if nr < 0 or len(self.board[nc]) <= nr:
42                break
```

```
43              if self.board[nc][nr] != p:
44                break
45            else:
46              return 1 if p == player else -1
47      return 0
48
49  other = {'X' : 'O', 'O' : 'X'}
50  player = 'X'
51  board = Board()
52
53  while True:
54    try:
55      c = int(input("%s -> " % player))
56    except TypeError:
57      continue
58    if not board.drop(player,c):
59      continue
60    print(board)
61    if board.score(player):
62      print("Player %s Wins!!!" % player)
63    elif board.full():
64      print("Tie")
65    else:
66      player = other[player]
67      continue
68    board = Board()
69    player = 'X'
70    print(board)
```

References

Fitzgerald, S., Lewandowski, G., McCauley, R., Murphy, L., Simon, B., Thomas, L., & Zander, C. (2008). Debugging: finding, fixing and flailing, a multi-institutional study of novice debuggers. *Computer Science Education*, *18*(2), 93–116.

Mosemann, R., & Wiedenbeck, S. (2001). Navigation and comprehension of programs by novice programmers. In *Program comprehension, 2001. iwpc 2001. proceedings. 9th international workshop on* (p. 79 -88). doi: 10.1109/WPC.2001.921716

Rajlich, V., & Wilde, N. (2002). The role of concepts in program comprehension. In *Program comprehension, 2002. proceedings. 10th international workshop on* (p. 271 - 278). doi: 10.1109/WPC.2002.1021348

von Mayrhauser, A., & Vans, A. (1995, aug). Program comprehension during software maintenance and evolution. *Computer*, *28*(8), 44 -55. doi: 10.1109/2.402076