Homework 03 - Project Revision

Andrew Carter and Beryl Egerter

February 14, 2013

1 Scheduled Interviews

Our scheduled interviews:

1. Interview with NB at 4:30pm on Saturday, February 16
2. Interview with NW at 9:30pm on Tuesday, February 19
3. Interview with DE at 9:00pm on Tuesday, February 26

2 Description of Topic

Often, whether in school or industry, programmers are given code that they have not seen before. They then must perform some task such as tracking down a bug or adding a feature. Familiarizing themselves with that code can take varying amounts of time. We would like to investigate how this process occurs and what factors affect it.

By learning how people reason about unfamiliar code and the results of their techniques we can help programmers be more prepared and confident in these situations. This skill is useful in real life situations, and is less abstract than targeting specific concept comprehension.

This topic is related to code comprehension for specific topics, such as recursion.

3 Empirical Goals

We are interested in observing the techniques that participants use to comprehend the code and make the required changes. Following are a couple of questions we can focus on while analyzing our interviews.

1. Does the participant look at what the function does or just the name, and to what extent does this affect success and confidence of correctness?
2. Did the participant even look at all of the code, or did they quickly (and correctly) narrow down the region that needed to be changed?
3. In what order did the participant explore the functions, and to what depth each time?

4 Response to Participant Questions

We realize that much of the code that we plan to ask participants to look at will be unfamiliar to them. Though we are interested in how they go about familiarizing themselves with the code, we do not intend to completely baffle them. Thus, we will answer any technical questions such as how the language work. We plan to answer any question that could be answered with a quick google search. We will not answer any questions about how the program is composed.

5 Non-Linear Interview Questions

Our questions are aimed to find a level that challenges the participant so that we can see the process more clearly. If a participant finds the Tier 1 Questions hard, we will not advance to Tier 2 Questions, and likewise for the Tier 2 Questions. We have prepared two questions each in Tier 1 and 2, and one question in Tier 3. Thus we can start with short, less complicated questions in Tier 1 and move up to Tier 3 which contains concepts the participant is unlikely to have seen if we believe the participant can handle it.

6 Interview Sequence and Questions

1. Welcome participant.
2. Read assent script to participant.
3. Solicit and answer any questions the participant has.
4. Have participant sign the consent form.
5. Provide participant a copy of the consent form.
6. Turn on the video camera.
7. Introduce topic: "We're very interested in how people look at unfamiliar code. Thus, for this interview we will be giving you pieces of code that you likely have not seen before and would like to observe you familiarizing yourself with it. To that end, we would like you to talk out loud about what you are looking at as much as possible. Most of the code will be provided on paper and there is a pencil that you can use as you work. We intend to observe your process and do not aim to completely baffle you. At the same time, we would like to see you familiarizing yourself with code that you might not immediately be able to tell what is going on. So, if you have questions about unfamiliar syntax or concepts we would be happy to answer them."
8. Have participant work on Question 1.1
   (a) Introduce question: "For this question we would like you to familiarize yourself with some Python code. Please explain to us what you think this code does."
   (b) Hand participant the attached handout, Question 1.1
   (c) Possible follow-up questions:
      i. Why did you start with this function?
      ii. Why did you move from this function to this other function?
      iii. What does the program end up doing?
9. If participant was not confident on Question 1.1, have participant work on Question 1.2
   (a) Introduce question: "For this question, we would like you to again familiarize yourself with some Python code. Please explain to us what you think this code does. "
   (b) Hand participant the attached handout, Question 1.2
   (c) Possible follow-up questions:
      i. Why did you start with this function?
      ii. Why did you move from this function to this other function?
      iii. What does the program end up doing?
10. If participant was confident enough on Question 1.1 or 1.2, have participant work on Question 2.1
    (a) Introduce question: "For this question we would like to have you look at some code in the programming language Ruby. This is the scenario: A coworker recently left on vacation and left two files behind. One of those files is a Date class and your boss wasn't sure if the coworker had finished including leap year support. Your boss would like you to make sure it is supported."
    (b) Hand participant the attached handout, Question 2.1
    (c) Possible follow-up questions:
       i. Why did you start here?
       ii. Why did you move from this function to this other function?
       iii. How did this feel different than looking at the Python code?
       iv. Ask about functions they should have been able to ignore and see to what extent they analyzed it.
11. If participant was not confident on Question 2.1, have participant work on Question 2.2
    (a) Introduce question: "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."
    (b) Hand participant the attached handout, Question 2.2

(c) Possible follow-up questions:

  i. Why did you start here?

  ii. Why did you move from this function to this other function?

  iii. How did this feel different than looking at the Ruby code?

  iv. How did this feel different than the first (or first and second) question?

  v. Ask about functions they should have been able to ignore and see to what extent they analyzed it.

12. If participant was confident enough on Question 2.1 or 2.2, have participant work on Question 3.1

  (a) Introduce question: ”For this question, we would like to have you look at some code in the programming language C. This is the scenario: The coworker who wrote the Date class also had built an RPN calculator, but your boss accidentally made a small change to the file, but he isn’t sure what he did, and now it won’t compile. Can you take a look at it, and see if you can find the change?”

  (b) Turn on screen capture.

  (c) Place computer with code before participant.

  (d) Possible follow-up questions:

    i. Why did you start here?

    ii. Why did you move from this function to this other function?

    iii. Ask about functions they should have been able to ignore and see to what extent they analyzed it.

    iv. Do they understand how the execution of the program works?

7 Handouts

Attached below.

Question 1.1

```
def func2(list, num):
    return func1(list, num, func4)

def func4(a, b):
    return a * b

def func1(list, num, f):
    acc = 0
    for i in list:
        acc += f(i, num)
    return acc

def main():
  print(func3([1,2,3,4]))

def func3(list):
  return func2(list, 4)

main()
```

Question 1.2

```
def function50(i, L):
  return L[i+2]

def function37(L):
  return [L[-1]]+L

def function52(i):
  return function4() * i

def function1(j, k):
  return (j + k) * function52(1)

def function4():
  return 3

def function188(L):
  return function37(L)+[function50(2, L)]

def function0():
  return function188([1,2,3,4,5,6,7,8,9])[function1(0,1)]

x = function0()
print x
```

Question 2.1, File 1

```
#!/usr/bin/ruby

load "ourdate.rb"

d = OurDate.new(2011,1,4)
print "#{d.what_day}"
print "We started writing this file today.\n"
d.forward_time(365)
print "We are almost done now.\n"
print "#{d.what_day}"
```

Question 2.1, File 2

```
#!/usr/bin/env ruby

$months31 = [1,3,5,7,8,10,12]
$months30 = [4,6,9,11]

class OurDate
  attr_accessor :year
  attr_accessor :month
  attr_accessor :day

  def initialize(year, month, day)
    @year = year
    @month = month
    @day = day
  end

  def is_equal?( d )
    puts @year == d.year and
      @month == d.month and
      @day = d.day
  end

  def is_leap_year?
    if @year % 400 == 0
      puts 1
    elsif @year % 100 == 0
      puts nil
    elsif @year % 4 == 0
      puts 1
    else
      puts nil
    end
  end

  def check_month
    if @month == 13
      @month = 1
      @year = @year + 1
    elsif @month == 0
      @month = 12
      @year = @year - 1
    end
  end

  def tomorrow
    @day = @day + 1
    if @day > 31
      for i in $months31
        if @month == i
          @day = 1
          @month = @month + 1
          check_month
        end
      end
    elsif @day > 30
      for i in $months30
        if @month == i
          @day = 1
          @month = @month + 1
          check_month
        end
      end
    elsif @day > 28 and @month == 2
      @day = 1
      @month = @month + 1
      check_month
    end
  end

  def yesterday
    @day = @day - 1
    if @day == 0
      @month = @month - 1
      check_month
      for i in $months31
        if @month == i
          @day = 31
        end
```

```ruby
      end
      for i in $months30
        if @month == i
          @day = 30
        end
      end
      if @month == 2
        @day = 28
      end
    end
  end

  def forward_time(n)
    for i in 0..n
      tomorrow
    end
  end

  def reverse_time(n)
    for i in 0..n
      yesterday
    end
  end

  def what_day
    puts "Today is #{month}/#{day}, #{year}!"
  end
end
```

Question 2.2

```python
#!/bin/env python3

class Board(object):
  def __init__(self, width=7, height=6):
    self.board = [[] for i in range(width)]
    self.width = 7
    self.height= 6
  def drop(self, player, column):
    if column < len(self.board):
      self.board[column].append(player)
      return True
    return False
  def __str__(self):
    result = ""
    for r in reversed(range(self.height)):
      result += "|"
      for c in range(self.width):
        if r < len(self.board[c]):
          result += self.board[c][r]
        else:
          result += " "
        result += "|"
      result += "\n"
    result += "-" * (2 * self.width + 1)
    return result
  def full(self):
    return all(len(col) >= self.height for col in self.board)
  def score(self, player):
    for c in range(self.height):
      for r in range(len(self.board[c])):
        p = self.board[c][r]
        for dc,dr in ((0,1),(1,0),(1,1),(1,-1)):
          for i in range(1,4):
            nc = c + i*dc
            nr = c + i*dr
            if nc < 0 or self.width <= nc:
              break
            if nr < 0 or len(self.board[nc]) <= nr:
              break
            if self.board[nc][nr] != p:
              break
          else:
            return 1 if p == player else -1
    return 0

other = {'X' : 'O', 'O' : 'X'}
player = 'X'
board = Board()
```

```
while True:
  try:
    c = int(input("%s > " % player))
  except TypeError:
    continue
  if not board.drop(player,c):
    continue
  print(board)
  if board.score(player):
    print("Player %s Wins!!!" % player)
  elif board.full():
    print("Tie")
  else:
    player = other[player]
    continue
  board = Board()
  player = 'X'
  print(board)
```