18-642 Recitation #3

September 13, 2019
Updates

• Homework:
  – Homeworks through #6 graded on canvas
  – Homeworks #9/10/11 due Wednesday night
• Homework grading
  – Points are mostly for good faith effort (no single “correct” answer)
  – Points taken off if:
    • You didn’t answer all parts of a question
    • You didn’t follow directions for a question
    • You didn’t cite/explain your work
    • You’re obviously phoning it in or being a jerk
  – Read comments on canvas, even if you got full points
  – Full points does not mean you got the right answer
    • We’ll try to cover some common issues in recitation
      If you’re not sure – ask!
Updates

• Projects:
  – Project 2 graded on canvas
  – Project 3 due tonight
  – Project 4 released, due in a week
Updates

• Reminder to use staff email and office hours for questions
  – Email only for administrative questions – it’s hard to answer conceptual questions
  – If you can’t make office hours, send us a note ahead of time with your availability and we’ll try to work something out
Today

• Project #4
• Homework discussion
Project #3 Questions?

• Reminder: you can omit conformance with any three checklist items
  – You have to tell us which three in writeup
  – For Project #4 you’ll have to fix them
  – Intended to give slack to students who are still coming up to speed on programming skills
Project 4

• Peer reviews
  – Your groups are assigned on canvas
  – There is a live lecture + video lecture on this

• For each review:
  – One person is the scribe
    • Person who has code under review
  – One person is the leader
    • Rotate this; by the end of all reviews, everyone has had a turn being scribe and being leader

• **Fix** any issues that come up **AFTER** the review
Peer Review Checklist

- Fill one out for **each** review

<table>
<thead>
<tr>
<th>Issue#</th>
<th>Issue Description</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Peer Review Reminders

- Inspect the item, not the author
- Don't get defensive
- Find but don't fix problems
- Limit meetings to two hours
- Keep a reasonable pace
- Avoid "religious" debates on style (While enforcing what the Project 3 checklist says)
What to review

• Go through code line by line
• Leader is in charge of
  – Saying when to move on to the next line
  – Reading through Project 3 style checklist out loud
• Prof Koopman’s review checklist is a good starting point for more general reviews:
  – https://betterembsw.blogspot.com/2018/01/
  – Note that some items won’t apply to your code
  – OK to use this too, but **Proj #3 style is mandatory**
Project #4 Questions?

• Group hand-in
  – All spreadsheets at end of review session
    • Status column will be blank

• Individual hand-in
  – Your code
  – Your spreadsheet with “status” filled in
  – A writeup
McCabe’s Cyclomatic Complexity

- Measures complexity by counting branching logic
- Generally used as a guideline to how readable/testable/maintainable source is
- High complexity of a module indicates:
  - Deeply nested loops
  - Lots of branch statements
  - Functionality that is not factored out into modules
- For detailed example, see posted recitation 3 slides online
Applications to testing

• Code coverage testing:
  – “White box” testing (can see the code)
  – Want to test all logic
  – Branch coverage testing
  – Path coverage testing
Branch coverage

Test cases
{x=1, y=2}
{x = 3, y=4}
All branches covered
Path coverage

Test cases:

{x=1, y=2}
{x=1, y=4}
{x=3, y=2}
{x=3, y=4}

All paths tested
Cyclomatic Complexity and Testing

# of tests to achieve full branch coverage
\[ \leq \text{MCC} \# \leq \]
# of tests to achieve full path coverage*

*Unless some paths are impossible

- Gives a lower bound to achieving path coverage (so a high MCC means a lot of testing)
- Also gives notion of how many test cases are needed to obtain “sufficient coverage”
- See McCabe paper, Section VII for more discussion


Subprocesses

for (i = 0; i < N; i++) {
    sub_process(i);
}

• Still has MCC of 2.
• Testing complexity in this module requires testing that the inside of the loop is reached, and testing the outside of the loop
• Complexity of sub_process() is computed separately because it can be unit-tested separately
• Takeaway: refactor code to reduce MCC
• Do this for project 3!!
MCC and SCC

• McCabe’s cyclomatic complexity
  – Counts # of if/while/for conditionals in the code
• Strict cyclomatic complexity
  – Includes +1 for every condition within a branch
• \( \text{if} \ (a < 0 \ \&\& \ b > 0) \) adds +1 to MCC, +2 to SCC
• You’ll encounter this in the homework
These have same MCC, but which is more readable?

if (isInBounds(x)) {
  if (y > 0) {
    if (prime factor(x, y)) {
      if (COMPUTE_FLAG) {
        do_function(x, y)
      }
    }
  }
}

if (!isInBounds(x)) {
  return false;
}
if (y <= 0) {
  return false;
}
if (!COMPUTE_FLAG) {
  return false;
}
do_function(x, y)
Spaghetti Code

- High MCC is just one indicator of spaghetti code
- Others:
  - Global variables
  - Copy-pasted code
  - Obfuscated variable names

background image: https://imgflip.com/memegenerator/Eminem
Questions?
Lightning Round

Which rule do you like & why?
Which rule do you hate and why?

• 1. Sections 1.2-1.5 (General Rules)
• 2. Sections 1.6-1.8 (General Rules)
• 3. Sections 2.1-2.2 (Comments)
• 4. Sections 3.1-3.3 (Whitespace Rules)
• 5. Sections 3.4-3.6 (Whitespace Rules)
• 6. Sections 4.1-4.4 (Module Rules)
• 7. Sections 5.1-5.5 (Data Type Rules)
• 8. Sections 6.1-6.3 (Procedure Rules)
• 9. Sections 7.1-7.2 (Variable Rules)
• 0. Sections 8.1-8.5 (Statement Rules)
Lightning Round Roster
HW Review: State Charts

- Used to express stateful behavior
- Number the states
- Name the states
- Input to system causes state transition
- Each state sets all output variables
- Avoid complex behaviors within state subroutine
- Avoid actions on transition

http://www.ece.cmu.edu/~ece642/lectures/08_modalstatechart.pdf
Bonus MCC/SCC Slides
Determining MCC

• Turn code into graph and count # of loops + 1

• Notation to turn code into graph:
  – Circle for statement(s) with no branching logic
  – Diamond for branching logic

http://www.ece.cmu.edu/~ece642/lectures/06_spaghettioode.pdf
Common constructions

• Sequential, no branching:

• IF statement:

• For loop:
Switch/multi-case IF:

```java
if (x == 1) {
    ...
} else if (x == 2) {
    ...
} else {
    ...
}
```

MCC = 3
IF within FOR:

for (i = 0; i < 10; i++) {
    if (i % 2 == 0) {
        ...
    } else {
        ...
    }
}

MCC = 3
Nested Loop:

```
for (i = 0; i < N; i++) {
    for (j = 0; j < N; j++) {
        ...
    }
}
```
More complex code

• Count total number of if/for/while statements and add 1
• Tools available to compute for you

https://betterembsw.blogspot.com/2014/06/avoid-high-cyclomatic-complexity.html
Pitfalls of MCC

- Not an exact # of test cases to run (testing generally strives for path coverage)
- Switch statements increase MCC a lot but are sometimes necessary for things like message translation
  - Prof Koopman’s blog post: https://betterembsw.blogspot.com/2014/06/avoid-high-cyclomatic-complexity.html
- Decent measure of code comprehension, but not the only one