CS 4/554, Introduction to Software Engineering
Winter 2020

Take-Home Exam
Due at 12PM, Thursday, Feb. 6, 2020

In preparing the answers for this examination, you may use any books, notes, or other generally available sources. You may NOT consult with anyone except the instructor about the examination. If you use an unusual source of information (for example, something cut and pasted from the Internet), please give a citation. Give a citation to the textbooks rather than copying material from them.

The exam should be submitted in hard copy on the due day.

Relative points assigned to each part are shown in brackets [ ].

1. [20 points] Here are four possible ways to use a pair of programmers in coding and unit testing a single subprogram from a given design:
   - **Share**: They work together on the programming and testing, sharing the work roughly equally.
   - **Separate**: They program independently, creating two versions of the code, which are run against each other for unit test.
   - **Inspect**: One does the programming, and the other inspects the resulting code for defects. There is no unit test.
   - **Test**: One does the programming, and the other does unit testing of the resulting code.

   Rank these, first: By how quickly the work is completed; and second: By the reliability of the resulting code. Justify your rankings briefly.

2. [15 points] In code inspections in a particular (hypothetical) company, one person is the moderator for all inspections. In order to evaluate which employees are the best inspectors, the moderator keeps track of the performance of each inspector, and employees know that this is being done. The moderator encourages dispute among inspectors in the inspection meeting, to determine which ones are more persuasive and better prepared. The moderator uses the collected information in assigning inspection duties, so that each inspection includes some inspectors with the best performance ratings. Explain what is wrong with the moderator’s actions.

3. [15 points] A path in the flow graph of a program is *infeasible* if it is impossible to execute that path in the program. Give an example program containing a single loop and no other conditional statements that has at least one infeasible path, and explain why this is so. What difficulty does the existence of infeasible paths raise for structural testing?
4. [30 points] Figure 1 presents a (buggy) example of QuickSort’s partition function. If you are not familiar with the quicksort algorithm, please refer to your algorithm book.

```c
void partition (int a[ ], int n) {
    int pivot = a[0];
    left = 0;
    right = n-1;

    while (left < right) {
        while (a[right] >= pivot) right--;
        if (left != right) {
            a[left] = a[right];
            left++;
        }

        while (a[left] < pivot) left++;
        if (left != right) {
            a[right] = a[left];
            right--;
        }
    }
    a[left] = pivot;
}
```

**Figure 1: A buggy partition function**

You are asked to test the function using the basis path testing introduced in the Appendix.

1. Draw the flow graph for the partition function.
2. Generate a basis set of independent paths that cover all the nodes and edges in the flow graph. (The definition of a basis set of independent path is given in the appendix.)
3. Generate a test case by specifying a[ ] and n so that the test case will trigger the bug in the partition function. Specify the path that is exercised by the test case.

5. [20 points] The path testing strategy can also be used to test state machine models. Please generate a basis set of independent paths that cover all the states and state transitions in the state machine model of the microwave oven given on page 142 of the textbook (or in the slides for Chapter 5). Each path should start and end in the “Waiting” state.
Appendix: Basis Path Testing

One of the biggest problems in unit testing is how to determine test cases. The tester has to ensure that there are enough tests to do thorough testing, but not so many tests that all of the limited testing time is used up. There are several different techniques to pick test cases including using boundary cases, dataflow testing and branch testing. No single technique can supply sufficient test cases, so usually a combination of techniques are used. This paper address one of those techniques, basis set testing. A basis set is a set of linearly independent paths that can be used to construct any path through the program flow graph.

Testing techniques can be divided into various categories. Black box testing ignores the internals of the software unit. Structured testing is, on the other hand, "Testing that takes into account the internal mechanism of a system or component". This can be further subdivided into different types including path testing, "Testing designed to execute all or selected paths through a computer program" and branch testing, "Testing designed to execute each outcome of each decision point in a computer program". Basis testing, also known as Structured Testing, is a hybrid between these two techniques. The test paths in a basis set fulfill the requirements of branch testing and also test all of the paths that could be used to construct any arbitrary path through the graph.

Any function in a program can be represented as a control flow graph. The nodes in this graph are program statements, while the directed edges are flow of control. Two nodes can be unconnected, connected by an edge in either direction, or connected by an edge in each direction. When tracing a path from the source to the sink, a back edge is a edge that leads back to a node that has already been visited. A flow graph contains one source node and one sink. A source node is a node which has no incoming edges, while a sink node is a node with no outgoing edges. A program function may have more than one sink, but this graph can be converted into a graph with only one sink. Some languages also allow more than one source. This construct is very rare and not used in Structured Programming.

A basis set is a set of linearly independent test paths. A path can be associated with a vector, where each element in the vector is the number of times that an edge is traversed. For example, consider a graph with 4 edges: a, b, c and d. The path ac can be represented by the vector [1 0 1 0]. Paths are combined by adding or subtracting the paths' vector representations. Each path in the basis set can not be formed as a combination of other paths in the basis set. Also, any path through the control flow graph can be formed as a combination of paths in the basis set.
This figure shows a simplified control flow graph. While a complete flow graph would not have two edges going to the same destination, this requirement has been relaxed to keep the number of paths to a manageable size for this example. A basis set for this graph is \{ac, ad, bc\}. The path bd can be constructed by the combination bc + ad - ac as shown in this table.

<table>
<thead>
<tr>
<th>Edge</th>
<th>bd</th>
<th>bc</th>
<th>bc + ad</th>
<th>bc + ad - ac</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>b</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>c</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>d</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

The set \{ac, bd\} is not a basis set, because there is no way to construct the path ad. The set \{ac, ad, bd\} is also a basis set. Basis sets are not unique; thus a flowgraph can have more than one basis set.

McCabe's complexity measure is a software metric that attempts to evaluate how complex a function is. The number of paths in the basis set is equal to the complexity measure of that function. The value of the complexity measure is equal to the cyclomatic complexity of the flow graph if all of the edges were undirected instead of directed. This can be calculated as equal to e - n + 2, where e is the number of edges and n is the number of nodes.