Midterm Review

CSCE 747 - Lecture 13 - 03/06/2018

We Will Cover

- You have a midterm on Thursday
  - 75 minutes
  - **Closed-book, closed-notes.**
  - Covers all content to date.
- There is a practice exam on the course site.
  - Did you try it?
- Let’s go over it!

Question 1

- A test suite that meets a stronger coverage criterion will find any defects that are detected by any test suite that meets only a weaker coverage criterion
  - True
  - False

- A test suite that is known to achieve Modified Condition/Decision Coverage (MC/DC) for a given program, when executed, will exercise, at least once:
  - Every statement in the program.
  - Every branch in the program.
  - Every LCSAJ in the program.
  - Every path in the program.

Question 1

- A test suite that meets a stronger coverage criterion will find any defects that are detected by any test suite that meets only a weaker coverage criterion
  - True
  - **False**

- A test suite that is known to achieve Modified Condition/Decision Coverage (MC/DC) for a given program, when executed, will exercise, at least once:
  - **Every statement in the program.**
  - **Every branch in the program.**
  - Every LCSAJ in the program.
  - Every path in the program.

Question 1

- Possible sources of information for functional testing include:
  - Requirements Specification
  - User Manuals
  - Program Source Code
  - Domain Experts

- Category-Partition Testing technique requires identification of:
  - Parameter characteristics
  - Representative values
  - Def-Use pairs
  - Pairwise combinations

Question 1

- Possible sources of information for functional testing include:
  - **Requirements Specification**
  - **User Manuals**
  - Program Source Code
  - **Domain Experts**

- Category-Partition Testing technique requires identification of:
  - **Parameter characteristics**
  - **Representative values**
  - Def-Use pairs
  - Pairwise combinations

Question 1

- Validation activities can only be performed once the complete system has been built.
  - True
  - False
- Statement coverage criterion never requires as many test cases to satisfy as branch coverage criterion.
  - True
  - False
- Requirement specifications are not needed for generating inputs to satisfy structural coverage of program code.
  - True
  - False
- A system that fails to meet its user's needs may still be:
  - Correct with respect to its specification.
  - Safe to operate.
  - Robust in the presence of exceptional conditions.
  - Considered to have passed verification.

Question 1

- Validation activities can only be performed once the complete system has been built.
  - True
  - **False**
- Statement coverage criterion never requires as many test cases to satisfy as branch coverage criterion.
  - True
  - **False**
- Requirement specifications are not needed for generating inputs to satisfy structural coverage of program code.
  - **True**
  - False
- A system that fails to meet its user's needs may still be:
  - **Correct with respect to its specification.**
  - **Safe to operate.**
  - **Robust in the presence of exceptional conditions.**
  - **Considered to have passed verification.**

Question 2

Consider the following situation:
After *carefully and thoroughly* developing a collection of requirements-based tests and running your test suite, you determine that you have achieved only 60% statement coverage. You are surprised (and saddened), since you had done a very thorough job developing the requirements-based tests and you expected the result to be closer to 100%.

Question 2

Briefly describe two (2) things that might have happened to account for the fact that 40% of the code was not exercised during the requirements-based tests.

- Poor job choosing test cases.
- Missing requirements.
- Dead or inactive code.
- Error-handling.
  - Code used only in special cases.

Question 2

Should you, in general, be able to expect 100% statement coverage through thorough requirements-based testing alone (why or why not)?

- No.
- There are almost always special cases not covered by requirements.
  - Code optimizations, debug code, exception handling.

Question 2

Some structural criteria, such as MC/DC, prescribe obligations that are impossible to satisfy. What are two reasons why a test obligation may be impossible to satisfy?

- Impossible combination of conditions
- Defensive programming (situations that may not happen in practice are planned for).
- Other situations that result in unused code (i.e., code implemented for future use that is not currently reachable).

Question 3

In class we discussed the importance of defining a test case for each requirement. What are the two primary benefits of defining this test case?

- Helps when performing integration testing. Can build test cases early, apply to code once it is written.
- Forces us to write testable requirements.

Question 4

The airport connection check is part of a travel reservation system. It checks the validity of a single connection between two flights in an itinerary.

```
validConnection(Flight arrivingFlight, Flight departingFlight)
                    returns ValidityCode.
```

A Flight is a data structure consisting of:

- A unique identifying flight code (string, three characters followed by four numbers).
- The originating airport code (three character string).
- The scheduled departure time (in universal time).
- The destination airport code (three character string).
- The scheduled arrival time (in universal time).

There is also a flight database, where each record contains:

- Three-letter airport code (three character string).
- Airport country (two character string).
- Minimum connection time (integer, minimum number of minutes that must be allowed for flight connections).

Question 4

A Flight is a data structure consisting of:

- A unique identifying flight code (string, three characters followed by four numbers).
- The originating airport code (three character string).
- The scheduled departure time (in universal time).
- The destination airport code (three character string).
- The scheduled arrival time (in universal time).

There is also a flight database, where each record contains:

- Three-letter airport code (three character string).
- Airport country (two character string).
- Minimum connection time (integer, minimum number of minutes that must be allowed for flight connections).

Identify categories and choices for the parameters of this function.

Question 4 - Solution

Parameter: Arriving flight

Flight code:
- *malformed*
- *not in database*
- *valid*

Originating airport code:
- *malformed*
- *not in database*
- *valid city*

Scheduled departure time:
- *syntactically malformed*
- *out of legal range*
- *legal*

Destination airport (transfer airport):
- *malformed*
- *not in database*
- *valid city*

Scheduled arrival time (tA):
- *syntactically malformed*
- *out of legal range*
- *legal*

Parameter: Departing flight

Flight code:
- *malformed*
- *not in database*
- *valid*

Originating airport code:
- *malformed*
- *not in database*
- *differs from transfer airport*
- *same as transfer airport*

Scheduled departure time:
- *syntactically malformed*
- *out of legal range*
- *before arriving flight time (tA)*
- *between tA and tA + minimum connection time (CT)*
- *equal to tA + CT*
- *greater than tA + CT*

Destination airport code:
- *malformed*
- *not in database*
- *valid city*

Scheduled arrival time:
- *syntactically malformed*
- *out of legal range*
- *legal*

Parameter: Database record

This parameter refers to the database time record corresponding to the transfer airport.

Airport code:
- *malformed*
- *not found in database*
- *valid*

Airport country:
- *malformed*
- *invalid*
- *valid*

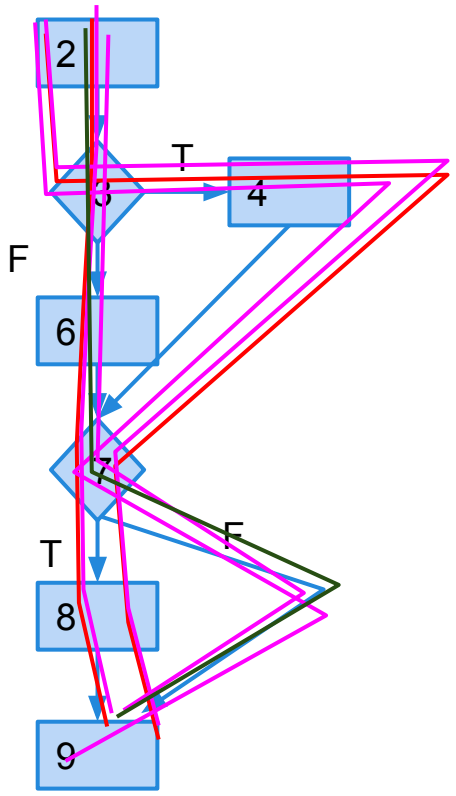
Minimum connection time:
- *not found in database*
- *invalid*
- *valid*

Question 5

- Draw the control-flow graph for this method.
- Develop test input that will provide statement coverage.
- Develop test input that will provide branch coverage.
- Develop test input that will provide path coverage.

```
int findMax(int a, int b, int c)
{
    int temp;
    if (a>b)
        temp=a;
    else
        temp=b;
    if (c>temp)
        temp = c;
    return temp;
}
```

Question 5 - Solution




```
1. int findMax(int a, int b, int c) {
2.   int temp;
3.   if (a>b)
4.        temp=a;
5.   else
6.        temp=b;
7.   if (c>temp)
8.        temp = c;
9.   return temp;
10. }
```

Statement:
(3,2,4), (2,3,4)

Branch:
(3,2,4), (3,4,1)

Path:
(4,2,5), (4,2,1), (2,3,4), (2,3,1)

Question 5 - Solution

- Modify the program to introduce a fault such that even path coverage could miss the fault.

Use (a >b+1) instead of (a>b) and the test input from the last slide: (4,2,5), (4,2,1), (2,3,4), (2,3,1) will not reveal the fault.

```
int findMax(int a, int b, int c)
{
    int temp;
    if (a>b)
        temp=a;
    else
        temp=b;
    if (c>temp)
        temp = c;
    return temp;
}
```

Question 6

- Identify all DU pairs and write test cases to achieve All DU Pair Coverage.
  - Hint - remember that there is a loop.

```
1. int doSomething(int x, int y)
2. {
3.     while(y > 0) {
4.         if(x > 0) {
5.             y = y - x;
6.         }else {
7.             x = x + 1;
8.         }
9.     }
10.    return x + y;
11. }
```

Question 6

```
1. int doSomething(int x, int y)
2. {
3.     while(y > 0) {
4.         if(x > 0) {
5.             y = y - x;
6.         }else {
7.             x = x + 1;
8.         }
9.     }
10.    return x + y;
11. }
```

Variable	Defs	Uses
x	1, 7	4, 5, 7, 10
y	1, 5	3, 5, 10

Variable	D-U Pairs
x	(1, 4), (1, 5), (1, 7), (1, 10), (7, 4), (7, 5), (7, 7), (7, 10)
y	(1, 3), (1, 5), (1, 10), (5, 3), (5, 5), (5, 10)

Question 6

```
1. int doSomething(int x, int y)
2. {
3.     while(y > 0) {
4.         if(x > 0) {
5.             y = y - x;
6.         }else {
7.             x = x + 1;
8.         }
9.     }
10.    return x + y;
11. }
```

Variable	D-U Pairs
x	~~(1, 4), (1, 5), (1, 7),~~ (1, 10), ~~(7, 4), (7, 5), (7, 7),~~ (7, 10)
y	~~(1, 3), (1, 5),~~ (1, 10), ~~(5, 3), (5, 5),~~ (5, 10)

Test 1: (x = 1, y = 2)
Covers lines 1, 3, 4, 5, 3, 4, 5, 3, 10

Test 2: (x = -1, y = 1)
Covers lines 1, 3, 4, 6, 7, 3, 4, 6, 7, 3, 4, 5, 3, 10

Test 3: (x = 1, y = 0)
Covers lines 1, 3, 8

Question 7

In a directed graph with a designated exit node, we say that a node **m** post-dominates another node **n**, if m appears on every path from n to the exit node.

Let us write *m pdom n* to mean that m post-dominates n, and *pdom(n)* to mean the set of all post-dominators of n, i.e., *{m | m pdom n}*.

Question 7

1. Does *b pdom b* hold true for all b?
2. Can both *a pdom b* and *b pdom a* hold true for two different nodes a and b?

1. Yes. Each node must appear on every path to the exit from itself.
2. Not if they are **different** nodes. If a pdom b, then b must be on all paths from a to the exit. Node a cannot appear after b at the same time.

Question 7

3. If both *c pdom b* and *b pdom a* hold true, what can you say about the relationship between c and a?
4. If both *c pdom a* and *b pdom a* hold true, what can you say about the relationship between c and b?

3. *c pdom a*. Node b appears on all paths from a to the exit. Node c must appear on the subpath from b to the exit.
4. Either *c pdom b* or *b pdom c*. Both b and c must appear on all paths from a to the exit. One will pdom the other.

Question 8

In class, we discussed various forms of oracles – such as a model, a second implementation, properties, self-checks, a team of experts, etc.

Provide a comparative analysis of three different kinds of oracles of your choice, defining what they are and addressing their strengths and weaknesses with respect to key attributes relevant to the verification process (e.g., cost, accuracy, completeness).

Question 8

- **Expected-Value Oracle:** Exact definition of the expected output given a concrete input. Most common form of oracle.

```
expected = 5;
actual = function(x);
assert(expected == actual);
```

- **Self-Check Oracle:** A property that must be met by the output, regardless of the value of the output.

```
actual = function(x);
assert (actual > 0);
```

- **Model:** A finite-state machine representing the abstract behavior of a function in a variety of situations.

Question 8

- Cost (per-test) (least to greatest):
  - expected value, self-check, model
  - Expected value very cheap, but only work for one input. Models very expensive, but can handle almost any input.
- Completeness (least to greatest):
  - expected value, self-check, model
  - Self-checks, models can account for more scenarios.
- Accuracy (least to greatest):
  - self-check, model, expected value
  - Self-checks only catch faults related to specified properties.

Question 9

- Explain the difference between *verification* and *validation*.
- Which of these is considered harder? Why?

Question 9

- Explain the difference between *verification* and *validation*.
  - Validation: Does the system meet the customer's needs? "Are we building the right product?"
  - Verification: Does the system meet the specifications we laid out? "Are we building the product right?"
- Which of these is considered harder? Why?
  - Validation is harder.
  - It requires that we understand the customer's actual desires. They might not have told us those, or changed their minds.

Question 10

Describe the key difference between black-box testing and white-box testing.

Question 10

Black-box testing treats the program as a machine that accepts input and issues output, with no visibility into its internal workings.

- Tests are based on requirements and specifications.
- You do not know what classes or methods are in the code, and you do now know what objects exist at runtime.

White-box involves testing the independent logic paths with full knowledge of the source code. You do not have full knowledge of the intended functionality (white box tests cannot look for unimplemented code).

Question 11

When we discuss software testing, we refer to Faults and Failures. Please briefly describe what a Fault is and what a Failure is. Make sure to point out the difference between a Fault and a Failure.

Question 11

- A Fault is a problem with the implementation. It is something that is missing, extra, or erroneous.
- A Failure is an incorrect execution of the software; we get an output we did not expect.
- A Failure is the manifestation of a Fault, if the execution executes the Fault and the corrupted state propagates to the output, we can observe it as a Failure.

Question 12

Why is it so important to include boundary values in your black-box test-data?

- Make sure your answer includes a brief description of what a boundary value is.

Question 12

Boundary values are the inputs that are on or close to the boundaries between the input equivalence partitions as well as special values we know are tricky to handle correctly.

- We know from experience that programmers make mistakes with boundary values.
- Thus we should include test cases to see if these cases are handled correctly.
  - Include values such as zero, very large, very small, empty list, max long list, etc.

Any other questions?

Next Class:

- The Midterm

Next Week:

- Spring Break

The Week After:

- Model-Based Test Creation