DIT635 - Practice Final

There are a total of 15 questions and 100 points available on the test. On all essay type questions, you will receive points based on the quality of the answer - not the quantity.

Make an effort to write legibly. Illegible answers will not be graded and awarded 0 points.

Question 1

1. A program may be correct, yet not reliable.
    a. True
    b. False

2. If a system is on an average down for a total 30 minutes during any 24-hour period:
    a. Its availability is about 98% (approximated to the nearest integer)
    b. Its reliability is about 98% (approximated to the nearest integer)
    c. Its mean time between failures is 23.5 hours
    d. Its maintenance window is 30 minutes

3. In general, we need either mock objects or drivers but not both, when testing a module.
    a. True
    b. False

4. If a temporal property holds for a finite-state model of a system, it holds for any implementation that conforms to the model.
    a. True
    b. False

5. Self-check oracles (assertions) do not require the expected output for judging whether a program passed or failed a test.
    a. True
    b. False

6. Object-oriented design and implementation typically have an impact on verification such that OO specific approaches are required for:
    a. Unit Testing
    b. Integration Testing
    c. System Testing
    d. Acceptance Testing

7. 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
    a. True
    b. False

8. 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:
    a. Every statement in the program.
    b. Every branch in the program.
    c. Every combination of condition values in every decision.
    d. Every path in the program.

9. Possible sources of information for functional testing include:
    a. Requirements Specification
    b. User Manuals
    c. Program Source Code
    d. Domain Experts

10. Category-Partition Testing technique requires identification of:
    a. Parameter characteristics
    b. Representative values
    c. Def-Use pairs
    d. Pairwise combinations

11. Validation activities can only be performed once the complete system has been built.
    a. True
    b. False

12. Statement coverage criterion never requires as many test cases to satisfy as branch coverage criterion.
    a. True
    b. False

13. Requirement specifications are not needed for selecting inputs to satisfy structural coverage of program code.
    a. True
    b. False

14. A system that fails to meet its user’s needs may still be:
    a. Correct with respect to its specification.
    b. Safe to operate.
    c. Robust in the presence of exceptional conditions.
    d. Considered to have passed verification.

Problem 2

Consider the software for air-traffic control at an airport (say, GOT). Air traffic control (ATC) is a service provided by ground-based air traffic controllers (the users of this system) who direct aircraft on the ground and through controlled airspace with the help of the software. The purpose of this software is to prevent collisions, organize and expedite the flow of air traffic, and provide information and other support for pilots.

The software offers the following features:

- Monitors the location of all aircraft in a user's assigned airspace.
- Communication with the pilots by radio.
- Generation of routes for individual aircraft, intended to prevent collisions.
- Scheduling of takeoff for planes, intended to prevent potential collisions.
- Alerts of potential collisions based on current bearing of all aircraft.
    - To prevent collisions, ATC applies a set of traffic separation rules, which ensure each aircraft maintains a minimum amount of empty space around it at all times.
    - The route advice can be either of "mandatory" priority (to prevent an imminent collision, pilots should follow this command unless there is a good reason not to) or "advisory" priority (this advice is likely to result in a safe route, but a pilot can choose to ignore it).

You may add additional features or make decisions on how these features are implemented, as long as they fit the overall purpose of the system. In any case, state any assumptions that you make.

Identify one performance, one availability, and one security requirement that you think would be necessary for this software and develop a quality attribute scenario for each.

Problem 3

Regarding performance:

1. What is the difference between response time and throughput?
2. Describe a situation where a system could display excellent throughput but poor response time and vice versa.

Question 4

You are building a web store that you feel will unseat Amazon as the king of online shops. Your marketing department has come back with figures stating that - to accomplish your goal - your shop will need an **availability** of at least 99%, a **probability of failure on demand** of less than 0.1, and a **rate of fault occurrence** of less than 2 failures per 8-hour work period.

You have recently finished a testing period of one week (seven full 24-hour days). During this time, 972 requests were served to the page. The product failed a total of 64 times. 37 of those resulted in a system crash, while the remaining 27 resulted in incorrect shopping cart totals. When the system crashes, it takes 2 minutes to restart it.

1. What is the rate of fault occurrence?
2. What is the probability of failure on demand?
3. What is the availability?
4. Is the product ready to ship? If not, why not?

Question 5

You are testing the following method:
```
public double max(double a, double b);
```

Devise four executable test cases for this method in the JUnit notation.

Question 6

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%.

1. 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.
2. Should you, in general, be able to expect 100% statement coverage through thorough requirements-based testing alone (why or why not)?
3. 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?

Question 7

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?

Question 8

The airport connection check is part of a travel reservation system. It is intended to check the validity of a single connection between two flights in an itinerary. For instance, if the arrival airport of Flight A differs from the departure airport of Flight B, the connection is invalid. Likewise, if the departure time of Flight B is too close to the arrival time of Flight A, the connection is invalid.

```
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 (string).
- Minimum connection time (integer, minimum number of minutes that must be allowed for flight connections).

`ValidityCode` is an integer with value 0 for OK, 1 for invalid airport code, 2 for a connection that is too short, 3 for flights that do not connect (arrivingFlight does not land in the same location as departingFlight), or 4 for any other errors (malformed input or any other unexpected errors).

In order to design requirements-based test cases, perform category-partition testing using this specification for the validConnection function.

1. Identify parameters.
2. Identify categories for each parameter.
3. Identify representative values (choices) for each category.

Question 9

For the following function,

a. Draw the control flow graph for the program.
b. Develop test input that will provide statement coverage. (Input output pairs will be fine.)
c. Develop test input that will provide branch coverage.
d. Develop test input that will provide full-path coverage.
e. Modify the program to introduce a fault so that you can demonstrate that even achieving full path coverage will not guarantee that we will reveal all faults. Please explain how this fault is missed in your example.

```
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 10

Identify all DU Pairs in the following code:

```
1.
2.  /* External file hex_values.h defined Hex_Values[128]
3.  * with value 0 to 15 for the legal hex digits
4.  * and value -1 for each illegal digit including special
5.  * characters */
6.
7.  #include “hex_values.h”
8.  /** Translate a string from the CGI encoding to plain
9.  * acsii text. ‘+’ becomes space, %xx becomes byte with hex
10. * value xx, other alphanumeric characters map to themselves
11. * Returns 0 for success, positive for erroneous input.
12. * 1 = bad hexadecimal digit.
13. */
14. int cgi_decode(char *encoded, char *decoded){
15.           char *eptr = encoded;
16.           char *dptr = decoded;
17.           int ok = 0;
18.           while(*eptr){
19.                  char c;
20.                  c = *eptr;
21.
22.                  if(c ==’+’){   /* Case 1: ‘+’ maps to blank*/
23.                         *dptr = ‘ ‘;
24.                  } else if(c == ‘%’){  /* Case 2: ‘%xx’ = char xx*/
25.                         int digit_high = Hex_Values[*(++eptr)];
26.                         int digit_low = Hex_Values[*(++eptr)];
27.                         if(digit_high == -1 || digit_low == -1){
28.                                /* *dptr=’?’ */
29.                                ok = 1; /* Bad return code */
30.                         }else{
31.                                *dptr = 16 * digit_high + digit_low;
32.                         }
33.                  }else{ /*Case 3: All other chars map to themselves*/
34.                         *dptr = *eptr;
35.                  }
36.                  ++dptr;
37.                  ++eptr;
38.           }
39.           *dptr = ‘\0’;
40.           return ok;
41.    }
```

Question 11

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}*.

Answer the following, providing justification for each:

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?
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?

Question 12

Consider the following function:

```
bSearch(A, value, start, end) {
        if (end <= start)
                return -1;
        mid = (start + end) / 2;
        if (A[mid] > value) {
                return bSearch(A, value, start, mid);
        } else if (value > A[mid]) {
                return bSearch(A, value, mid+1, end);
        } else {
                return mid;
        }

}
```

Give an example, with a brief justification, for each of the following kinds of mutants that may be derived from the code by applying mutation operators of your choice. Do not reuse a mutation, even if it fits multiple categories.

1. Equivalent Mutant
2. Invalid Mutant
3. Valid, but not Useful
4. Useful Mutant

Question 13

Suppose that finite state verification of an abstract model of some software exposes a counter-example to a property that is expected to hold for true for the system. Briefly describe what follow-up actions would you take and why?

Question 14

Temporal Operators: A quick reference list.

- G p: p holds globally at every state on the path
- F p: p holds at some state on the path
- X p: p holds at the next (second) state on the path
- p U q: q holds at some state on the path and p holds at every state before the first state at which q holds.
- A: for all paths from a state, used in CTL as a modifier for the above properties (AG p)
- E: for some path from a state, used in CTL as a modifier for the above properties (EF p)

Consider a finite state model of a traffic-light controller similar to the one discussed in the homework, with a pedestrian crossing and a button to request right-of-way to cross the road.

State variables:

- **traffic_light: {RED, YELLOW, GREEN}**
- **pedestrian_light: {WAIT, WALK, FLASH}**
- **button: {RESET, SET}**

Initially: **traffic_light = RED, pedestrian_light = WAIT, button = RESET**

Transitions:
pedestrian_light:

- **WAIT → WALK if traffic_light = RED**
- **WAIT → WAIT otherwise**
- **WALK → {WALK, FLASH}**
- **FLASH → {FLASH, WAIT}**

traffic_light:

- **RED → GREEN if button = RESET**
- **RED → RED otherwise**
- **GREEN → {GREEN, YELLOW} if button = SET**
- **GREEN → GREEN otherwise**
- **YELLOW→ {YELLOW, RED}**

button:

- **SET → RESET if pedestrian_light = WALK**
- **SET → SET otherwise**
- **RESET → {RESET, SET} if traffic_light = GREEN**
- **RESET → RESET otherwise**

1. Briefly describe a safety-property (nothing “bad” ever happens) for this model and formulate it in CTL.
2. Briefly describe a liveness-property (something “good” eventually happens) for this model and formulate it in LTL.
3. Write a trap-property that can be used to derive a test case using the model-checker to exercise the scenario “pedestrian obtains right-of-way to cross the road after pressing the button”.

Question 15

Consider a simple microwave controller modeled as a finite state machine using the following state variables:

- Door: {Open, Closed} -- sensor input indicating state of the door
- Button: {None, Start, Stop} -- button press (assumes at most one at a time)
- Timer: 0...999 -- (remaining) seconds to cook
- Cooking: Boolean -- state of the heating element

Formulate the following informal requirements in CTL:

1. The microwave shall never cook when the door is open.
2. The microwave shall cook only as long as there is some remaining cook time.
3. If the stop button is pressed when the microwave is not cooking, the remaining cook time shall be cleared.

Formulate the following informal requirements in LTL:

1. It shall never be the case that the microwave can continue cooking indefinitely.
2. The only way to initiate cooking shall be pressing the start button when the door is closed and the remaining cook time is not zero.
3. The microwave shall continue cooking when there is remaining cook time unless the stop button is pressed or the door is opened.