Written Examination
DIT636/DAT560 – Software Quality and Testing
March 13, 2024; 8:30 – 12:30

Course Responsible/Examiner: Gregory Gay
E-Mail: ggay@chalmers.se
Phone: +46 73 856 77 93
Examination Hall Visits: 9:30, 11:30
Allowed aids: No notes or other aids are allowed.
Grading Scale: 0-49 (U), 50-69 (3), 70 - 85 (4), 86-100 (5)
Examination Review: To Be Announced on Canvas

There are a total of 9 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. Illegible answers will not be graded.

Question 1 (Warm Up) - 10 Points

Multiple solutions may apply. Select all that are applicable.

1. For the expression `(a || c) && (b && !c)`, the test suite **(a, b, c) = {(T, F, T), (F, F, T), (F, T, T), (T, F, F)}** provides:
   a. MC/DC Coverage
   b. Decision Coverage
   c. Basic Condition Coverage
   d. Compound Condition Coverage

2. The supermodel tour can be used to check that the user manual is in alignment with the application-under-test.
   a. True
   b. False

3. MC/DC coverage subsumes branch coverage.
   a. True
   b. False

4. A typical distribution of test types is 70% unit tests, 20% system tests, and 10% GUI/exploratory tests.
   a. True
   b. False

5. In a genetic algorithm, which of the following operators are employed to create a new population?
   a. Crossover
   b. Cooldown
   c. Tournament Selection
   d. Mutation

6. An equivalent mutant is the same thing as an invalid mutant.
   a. True
   b. False

7. You are designing a chatbot for your website to help answer customer questions. Which one of the following quality attributes is the most important?
   a. Availability
   b. Performance
   c. Scalability

   Briefly (1-2 sentences), explain why this is the most important.

Question 2 (Quality Scenarios) - 10 Points

Consider a web-based system for advertising events. This system:

- Offers a list of upcoming events, which can be filtered by a date range or type of event.
- A user can click on an event to display detailed information about the event, such as its date, time, location, ticketing information, and a description. A user can also share a link to an event to a social network (e.g., Twitter or Instagram).
- Event organizers can create an account and log in to submit information on new events. They must pay a fee to post an event. They can also edit details of their events and see metadata such as the number of users who opened, saved, or shared an event.
- Users can create an account and log in to save events for later.
- The account details for both types of accounts can be managed, including the user's password, address, and e-mail address. Event organizers can also store a credit card number for making payments.

This service must provide functionality in a high quality manner to thousands of concurrent visitors.

Create one **performance** and one **availability** scenario for this system, with a Description, System State, System Environment, External Stimulus, Required System Response, and Response Measure for each.

Question 3 (Testing Concepts) - 10 Points

Choose one of the stages of testing that we covered: unit, system, or exploratory testing.

Explain in your own words what that stage is, how systems are tested in that stage, how it differs from the other listed stages of testing, and the types of faults that are most likely to be exposed by that stage that would be missed during the other stages.

Question 4 (System Testing) - 12 Points

Consider an employee management program that offers an API where, among other functions, a user can apply for a transfer to a new job:

```
public boolean applyForTransfer (String userID, String jobID)
```

The function returns TRUE if the user successfully applied for the transfer (i.e., they meet all qualifications and are eligible). It returns FALSE if not. An exception can also be thrown if there is an error.

This function connects to two databases, a user database, and a job database. Each user has the following items stored in their database entry, all in string format:

- User ID (expected format: "firstname.lastname")
- Current role
- Current salary
- Time in the current role
- Highest degree earned
- Citizenship status

Each job has the following items stored in its database entry:

- Job ID
- Minimum current salary to apply
- Maximum current salary to apply
- Minimum time in current role to apply
- Degree required to apply
- Citizenship status required to apply
- An array of roles that are allowed to transfer to this job (if empty, any current role is allowed, as long as other requirements are met)

Perform category-partition testing for this function.

1. Identify choices (controllable aspects that can be varied when testing)
2. For each choice, identify representative input values.
3. For each value, apply constraints (IF, ERROR, SINGLE) if they make sense.

You do not need to create test specifications or concrete test cases. For invalid input, **do not** just write "invalid" - be specific. If you wish to make any additional assumptions about the functionality of this method, state them in your answer.

Question 5 (Exploratory Testing) - 8 Points

Exploratory testing typically is guided by "tours". Each tour describes a different way of thinking about the system-under-test and prescribes how the tester should act when they explore the functionality of the system.

1. Describe one of the tours that we discussed in class other than the supermodel tour.
2. Consider the event management system from Problem 2. Describe three sequences of interactions with one or more functions of the system you would explore during exploratory testing of this system, based on the tour you described above. Explain the interactions you would take when executing that sequence, and why those actions fulfill the goals of that tour.

Question 6 (Unit Testing) - 8 Points

Consider a Java method that takes in a string and returns the specified substring:

```
public string mySubstring(String word, int start, int end);
```

Write JUnit-format test cases to do the following:

1. Create a test case that checks a normal usage of the method.
2. Create a test case reflecting either an error-handling scenario OR a performance scenario.

Question 7 (Structural Testing) - 15 Points

This function finds the longest increasing continuous subsequence in a given array of integers.

```
1. public static int longest_seq(int[] nums) {
2.     int max_sequ = 0; // Initializing the maximum sequence length
3.     // Handling the case when the array contains only one element
4.     if (nums.length == 1)
5.         return 1;
6.     for (int i = 0; i < nums.length - 1; i++) {
7.         int ctr = 1; // Counter to track the sequence length
8.         int j = i; // Initializing j to the current index i
9.         // Checking for an increasing sequence
10.        if (nums[i + 1] > nums[i]) {
11.            while (j < nums.length - 1 && nums[j + 1] > nums[j]) {
12.                ctr++; // Incrementing for each increasing element
13.                j++;
14.            }
15.        }
16.        // Updating the maximum sequence length encountered so far
17.        if (ctr > max_sequ)
18.            max_sequ = ctr;
19.        // Move the index by sequence length - 2 to avoid rechecking
20.        i += ctr - 2;
21.    }
22.    return max_sequ;
23. }
```

For example:
`longest_seq([10, 11, 12, 13, 14, 7, 8, 9, 1, 2, 3])` returns 5
`longest_seq([4, 5, 12, 13, 14, 7, 8, 9, 1, 2, 3])` also returns 5

1. Draw the control-flow graph for this program. You may refer to line numbers instead of writing the full code.
2. Identify test input that will provide statement and branch coverage. You do not need to create full unit tests, just input for Strings s1 and s2. For each input, list the line numbers of the statements covered as well as the branches covered (use the line number and T or F, i.e., "3-T" for the true branch of line 3).

Question 8 (Mutation Testing) - 15 Points

This method searches for a specified integer in a sorted array of integers.

```
1. public static int jumpSearch(int[] nums, int item) {
2.     int array_size = nums.length;
3.     // Find block size to be jumped
4.     int block_size = (int)Math.floor(Math.sqrt(array_size));
5.     // Find the block where element is present
6.     int prev_item = 0;
7.     while (nums[Math.min(block_size, array_size)-1] < item) {
8.         prev_item = block_size;
9.         block_size += (int)Math.floor(Math.sqrt(array_size));
10.        if (prev_item >= array_size)
11.            return -1;
12.    }
13.    // Linear search for element in block beginning with prev. item
14.    while (nums[prev_item] < item){
15.        prev_item++;
16.        if (prev_item == Math.min(block_size, array_size))
17.            return -1;
18.    }
19.    // If element is found
20.    if (nums[prev_item] == item)
21.       return prev_item;
22.    return -1;
23. }
```

Answer the following three questions for **each** of the following mutation operators:

- Relational operator replacement (ror)
- Arithmetic operator replacement (aor) (including short-cut operators)
- Constant for constant replacement (crp)

1. Identify all lines that can be mutated using that operator.
2. Choose **one** line that can be mutated by that operator and create **one** non-equivalent mutant for that line.
3. For that mutant, identify test input that would detect the mutant. Show how the output (return value of the method) differs from that of the original program.

Question 9 (Finite State Verification) - 12 Points

Temporal Operators: A quick reference list. p is a Boolean predicate or atomic variable.

- G p: p holds globally at every state on the path from now until the end
- F p: p holds at some future state on the path (but not all future states)
- X p: p holds at the next 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 reaching out from a state, used in CTL as a modifier for the above properties (AG p)
- E: for one or more paths reaching out from a state (but not all), used in CTL as a modifier for the above properties (EF p)

An LTL example:

- G (MESSAGE_SENT -> F (MESSAGE_RECEIVED))
- It is always true (G), that if the message is sent (property MESSAGE_SENT is true), then at some point after it is sent (F), the message will be received (property MESSAGE_RECEIVED will become true).
    - More simply: A sent message will always be received eventually.

A CTL example:

- EG (WIND -> AF (RAIN))
- There is a potential future where it is a certainty (EG) that - if there is wind (property WIND is true) - it will always be followed eventually (AF) by rain (property RAIN will become true).
    - More simply: There is some probability that wind will inevitably lead to eventual rain, but we have not established this fact for certain.

Consider a warehouse system where a robot prepares shipments for delivery to a customer. A finite state model of this system has the following state variables:
Order_Status: {Placed, Packaging, Shipped}
Robot_Status: {Idle, Working}
Robot_Arm_Status: {Idle, Moving, Grabbing, Holding, Releasing}
Robot_Leg_Status: {Idle, Moving, Standing}
Packaging_Status: {No Box, Box Requested, Box Closed, Box Open, Box Sealed}

1. Write two safety properties (something "bad" must never happen) for this model and formulate them in LTL or CTL. These properties must exercise different scenarios, and not just have variable names changed.

2. Write two liveness properties (something "good" eventually happens) for this model and formulate them in LTL or CTL. These properties must exercise different scenarios, and not just have variable names changed.

Explain each property. You must use LTL for at least one property, and you must use CTL for at least one property, otherwise you may choose between them.