Written Examination
DIT635 – Software Testing and Analysis
June 8, 2021 8:30 – 12:30

Name:

Course Responsible/Examiner: Gregory Gay
E-Mail: ggay@chalmers.se
Phone: Not used
Examination Hall Visits: Not applicable
Allowed aids and additional information: You may refer to the textbook, slides, and other course materials. However, do not quote directly from these materials (answer in your own words). You may not collaborate with others.

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. You may answer either in this document or in a separate document.
Grading Scale: 0-49 (U), 50-85 (G), 86-100 (VG)
Examination Review: Online (via e-mail)

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. Acceptance testing is a critical validation activity.
    a. True
    b. False

3. All DU pairs coverage requires that all paths between each definition and each of its usages be covered by at least one test case.
    a. True
    b. False

4. A banking website that displays an error message and stops all normal operations when a database connection cannot be established is which of the following:
    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.

5. In random ascent, we take the first neighbouring solution to show any improvement over the current solution as the new solution.
    a. True
    b. False

6. If all mutants are detected by a test suite, we have proof that all real faults have been detected as well.
    a. True
    b. False
7. You are designing a navigation app for mobile phones. You have designed the app to resend information if an update request fails. What reliability measures would be of most interest to you?
    a. Availability
    b. Probability of failure on demand
    c. Mean time between failures
    d. Rate of fault occurrence

Question 2 (Quality Scenarios) - 10 Points

Consider an online video streaming site, like Netflix. This site allows users to search and browse titles, save them to playlists, and stream them on demand. This site must serve high-definition video to thousands of concurrent users around the world.

Identify one performance and one availability requirement that you think would be necessary for this system and develop a quality scenario to verify each of them, 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 stage of testing (unit testing, system testing, GUI testing, exploratory testing, or acceptance 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 (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 a web-based education management system like LADOK, where one can register for courses, see past courses taken, apply for credits or graduation, manage their student information, apply for a university, or transfer to a new university. Describe three actions you might take during exploratory testing of this system, based on the tour you described above.

Question 5 (Unit Testing) - 8 Points

Consider a Java method for converting a date, expressed in String form (“YYYY-MM-DD”) to an instance of a CustomDate class (CustomDate has integer-typed fields Year, Month, and Day).

```
CustomDate convertToDate (String toConvert) throws IllegalFormatException;
```

For example, `convertToDate(“2020-05-01”)` returns a CustomDate object containing the fields (year=2020, month=5, day=1).

e.x. `if (CustomDate.year == 2020) { System.out.println(“It’s 2020”); }`

Write JUnit-format test cases to do the following:

1. Check that a well-formatted and legal date string returns the expected field values from the CustomDate object.
2. Check that a well-formatted, but illegal date results in the method throwing a `IllegalFormatException` (for example, “2020-31-12” is illegal because there is no 31st month of the year).

Question 6 (Structural Testing) - 15 Points

For the following method:

```
1. public int largestPrimeFactor(int n) {
2.    int factor = -1;
3.    for (int i = 2; i * i <= n; i++) {
4.       if (n % i != 0) {
5.           continue;
6.       }
7.       factor = i;
8.       while (n % i == 0) {
9.           n /= i;
10.       }
11.   }
12.   if (n == 1) {
13.       return factor;
14.   } else{
15.       return n;
16. }
```

1. Draw the control-flow graph for this method. You may refer to line numbers instead of writing the full code.
2. Develop test input that will provide statement and branch coverage. For each test, 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).
3. Do your test cases also achieve path coverage? Briefly explain your answer.

Question 7 (Data Flow Testing) - 12 Points

This method computes the longest common sequence of characters between two strings:

```
1. public String findLongestCommonSequence(String s1, String s2) {
2.    String result = "";
3.    for (int length = s1.length(); length > 0; length--) {
4.        int startIndex = 0;
5.        while (startIndex + length <= s1.length()) {
6.            String current = s1.substring(startIndex, startIndex + length);
7.            if (s2.contains(current)) {
8.                result = current;
9.                break;
10.            }
11.            startIndex++;
12.       }
13.       if (result.length() != 0) {
14.           break;
15.       }
16.   }
17.   return result;
18. }
```

1. Identify the def-use pairs for all variables.
2. Provide a test suite that achieves all def-use pairs coverage.

Question 8 (Mutation Testing) - 15 Points

Consider the following function:

```
1. public int sum(int n) {
2.     int sum = 0;
3.     for (int i = 1; i <= n; i++) {
4.         if (i % 3 == 0 || i % 5 == 0) {
5.             sum += i;
6.         }
7.     }
8.     return sum;
9. }
```

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

- Relational operator replacement (ror)
- Arithmetic operator replacement (aor)
- 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.
3. For that mutant, provide 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.

- 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 (for example, AG p)
- E: for some path from a state, used in CTL as a modifier for the above properties (for example, EF p)

Consider a simple elevator. A finite state model of this system has the following state variables:

Button_Floor1: {REQUESTED, NOT_REQUESTED}
Button_Floor2: {REQUESTED, NOT_REQUESTED}
Button_Floor3: {REQUESTED, NOT_REQUESTED}
Button_Floor4: {REQUESTED, NOT_REQUESTED}
Cabin_Floor: {1,2,3,4}
Cabin_Direction: {NOT_MOVING, MOVING_UP, MOVING_DOWN}
Cabin_Door: {Open, Closed}

1. Write three 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.