On Basic Testing Terminology, Dependability, and Testing Principles
QUIZ

1. Why is testing considered an 'optimistic' verification technology?
- Any tests that fail may not actually matter from the user's perspective
- Some tests may fail, but the program may be correct.
- [✓ The tests may all pass but the program may still be incorrect]
- You can use testing on any kind of software, whereas you can only run static analysis or proofs on small systems.

2. If a test fails, it could mean:
- [✓ The program is incorrect]
- [✓ The test itself is incorrect]
- There is no problem

3. Suppose we have two systems A and B, and A is more rigorously tested than B. Does System A better meet its quality goals than System B?
- Yes
- [✓ No]

4. Unit test is used to test: (choose best answer)
- User initial requirements.
- All of the above.
- [✓ Implementation of the software.]
- Design of the software.

5. 'Service' is the system behavior as defined by the software requirements.
- [✓ False]
- True

'Service' is the behavior as perceived by the user of the system. This behavior may or may not match the requirements.

6. A latent error becomes an effective error when
- [✓ the program reaches a state where the error manifests.]
- the user types it into the code
- the error causes the program to return something unexpected to the user.
- the program starts executing
- the program executes the line of code containing the latent error.

7. Does a program terminating with an error always indicate a failure?
- Yes
- [✓ No]

There are many reasons for error termination. For example, if the program cannot continue because erroneous inputs would render the result misleading or incorrect, it is often part of the service specification to terminate with error.

8. Adaptive cruise control software that continues to run in the presence of multiple hardware and service failures but regularly misjudges the distance between cars by a substantial amount is an example of a

- certainly robust
- certainly safe
- certainly reliable
- certainly correct
- certainly incorrect

9. Mutation testing is a __________ metric

- white-box
- black-box
- gray-box

Mutation testing is a white-box metric: it examines the syntax of the program and makes small changes (mutations) to the code. It does not use the requirements or program inputs in the computation of the metric, as would occur in black-box and/or grey-box metrics.

10. Which of the following are true about testing? Check all that apply.

- It is difficult to do rigorously. *{Due to discontinuities with software and the vast number of states, software testing is very difficult to do well.}*
- It checks the whole system, including software that you didn't write. *{Tests should check all the code in your system and are much more useful for software you did not write.}*
- It documents system behavior. *{The expected output of a function given an input can easily be seen through testing.}*
- It can sometimes find errors that are not actively looked for, when (for example) a program crashes during execution of a test case. *{Many errors are found by running a test that are not in fact the goal of the test.}*
- It can conclusively determine whether the software is correct.

11. Tasks that can be part of the Tear Down phase are (choose two answers):

- Initialize test case values.
- Remove data you added after testing is done.
- Open connection for testing.
- Close connection after testing is done.

12. In the JUnit test framework, we write test cases (choose the best answer):

- inside the executed method and we annotate that this part is for testing.
- inside the class to be tested we annotate that this part is for testing.
- in a separate class, and for each method in the program we associated a test case(s) to test the correctness of the method.
- All of the above.

{That is right! You need to have a separate class for testing that maps to each class in the program. The testing class will contain testing methods that test each method success and corner cases.}

13. Concurrent systems are quite difficult to test because (choose the best answer):

- the different potential interleaving of threads leads to 'race conditions' where the program may behave differently between executions
- All of the above
- to ensure that only one thread uses a variable at a time, Java 'synchronized' code can lead to deadlocks where each thread is blocked waiting for another thread.
- the different potential interleavings of threads means that there are many more possible system states