Test Driven Development

Name: _____ ID: _____

These questions do not have a formal, definitive answer. They are meant to be food for thoughts. Feel free to seek answers on browsing the Internet, talking to other software developers or reading books.

1. Many software engineering process follow the *spiral model* in which the development phases are divided into *specification*, *design* and *coding*. For each of these phases there are testing steps that should be performed: *unit testing*, *integration tests* and *validation tests*. What is the correspondence between testing phases and development phases?

2. When can we start testing the software? When can we stop testing it?

3. Who is in charge of testing the software?

4. Some software development companies hire independent testing teams. What are the advantages and disadvantages of this approach?

5. Testing gives much more feedback that the ordinary "the program works" and "the program does not work". Which information is worth keeping from the results of testing steps?

6. Unit testing probes the smallest compilable units in the program. Which aspects of the system can be tested in this way?

7. Incorrect conversion of strings to data types is an error commonly found during unit testing. For instance, `int c = Integer.parseInt(s)` may produce an exception if `s` does not represent a number. Give some examples of other errors that might be found by unit tests.

8. Unit tests, in general, are performed on a program module, when the whole program has not been finished yet. How is that possible?

9. Integration tests are performed to find errors when we combine modules to build bigger parts of the program. However, given that all the modules have been unit tested, what could go wrong when we put them together? Worse: if we put the modules together, and something goes wrong, how to find out the error?

10. There are two main approaches to perform integration tests: *top down integration* and *bottom up integration*. What are the steps taken in each of these test methodologies?

11. Both integration test methodologies, the top down approach, and the bottom up approach are *incremental*. This means that a sub-component is tested, and then a new module is added to the tested component, and a new test is performed. These new tests are called *regression tests*. How to organize regression tests? Should we perform all the old tests again? Should we add some new tests? Are there tests that should be emphasized?

12. *Validation tests* are used to find flaws in the software, in the context in which the software is used. The focus is on the requirements, that is, on the part of the system that the end user will see. In your own words, what is the difference between validation and verification?

13. How to know if a program meets the requirements that gave origin to its design?

14. There are two types of validation tests: *alpha validation* and *beta validation*. Define each of these concepts, and explain how each testing phase is performed.

15. Unit tests, integration tests, validation tests, ... are there more to test?

16. The final product must be tested in face of exceptional conditions. These are called *system tests*. Give examples of such exceptional conditions.

17. Tests will inevitably lead to bugs, which must be removed. Why debugging is such a difficult art? List a number of factors that complicate debugging.

18. There are different approaches to debugging. Examples include *brute force*, *backtracking* and *cause elimination*. Describe each approach.

19. Wonderful: the bug has been found. Are there other measures that the development team should take, besides removing the bug?

20. Let's do a bit of test guided development. In order to do this, you should develop a small application that will keep track of results of the National Soccer Championship. I believe that a reasonable program would have to pass at least the test sequence below:

```
import junit.framework.*;
public class TestCampaign extends TestCase {
  public void testTeamCreation() {
    Campaign c = new Campaign();
    c.addResult("Flamengo", "Fluminense", 2, 1);
    assertEquals(c.getPoints("Flamengo"), 3);
    assertEquals(c.getPoints("Fluminense"), 0);
    assertEquals(c.getGoalsPro("Fluminense"), 1);
    assertEquals(c.getGoalsCon("Fluminense"), 2);
  }
  public static void main(String args[]) {
    junit.textui.TestRunner.main(new String[] {"TestCampaign"});
  }
}
```

You must code the `Campain` class, and then augment the `TestCampain` class to do some further testing in your application.