Hyperiondev

TASK

Software Testing

Visit our website

Introduction

WELCOME TO THE SOFTWARE TESTING TASK!

All software developers need to be able to ensure the quality of their code. Software testing is essential in this regard! This task will explain the aims of software testing as well as some of the most important types of testing that should be done.

Get in touch

Connect for support

Remember that with our courses, you're not alone! You can contact your mentor to get support on any aspect of your course.

The best way to get help is to login to **www.hyperiondev.com/portal** to start a chat with your mentor. You can also schedule a call or get support via email.

Your mentor is happy to offer you support that is tailored to your individual career or education needs. Do not hesitate to ask a question or for additional support!

WHAT IS TESTING?

Testing something is a concept we're all familiar with, but not many of us have thought a lot about. Say you were thinking about buying a car, what would be included in your test criteria? It could include requirements like: is it an easy car to drive? Has it passed all the roadworthy inspections? Is it comfortable? It could also include requirements like: how safe is the car? Does it have ABS and traction control? Finally, how do we know when we have found the ideal car?

Without even realising it, when testing a car before we buy it, we are actually doing the two main types of testing we do in programming too! The first is to see if the car does what it's meant to do (i.e. can we drive to work and home again without it breaking down). This is called **validation testing**. The second is to see if, in an extraordinary circumstance, the car will still be okay (i.e. if we have to slam on brakes, we want the ABS to kick in so the car doesn't skid). This is known as **defect testing**.

To put it in programming terms, **validation testing** "demonstrate[s] to the developer and the customer that the software meets its requirements" (Sommerville, 2011, p. 206), while **defect testing** "discover[s] situations in which the behaviour of the software is incorrect, undesirable, or does not conform to its specification [...] test cases in defect testing can be deliberately obscure and need not reflect how the system is normally used" (Sommerville, 2011, p. 206).

Now, look back to our final question, "How do we know when we have found the ideal car?" The truth is, because we can't test all the cars in the world, we will never truly know for sure. The best we can do is find a car that fulfils our specific requirements. In programming, we have the same issue: testing is not exhaustive. To put it simply: **"Testing can only show the presence of errors, not their absence."** (Dijkstra et al., 1972, as cited in Sommerville, 2011, p. 206).

SOFTWARE TESTING

Before we get into the details of software testing, let's first look at some terminology:

- **Test data:** These are inputs that are used to test the system. They can be manually or automatically generated.
- **Test cases:** These are manually generated cases that specify the expected results of the test. It is what the actual results of the tests are then compared with.

- **Test execution:** This is the automated process of running the test and comparing the expected results to the generated results.

During the software testing process there are three general stages of testing:

1. Development testing
2. Release testing
3. User testing

Each of these will be discussed in detail below.

DEVELOPMENT TESTING

As the name suggests, development testing is the process of finding errors during the development process. This means it is usually done by system designers and programmers. This stage of testing is mostly defect testing, e.g. finding bugs in the system.

You may remember from *Agile Development* that pairs of programmers can work together to test and develop increments of a system. In some environments, however, a programmer may be paired with a designated tester that creates tests for the programme as it is being written. On the other hand, there could be a full testing team whose express purpose is to test the system during this phase. It is their responsibility, then, to create the tests and keep records of the outcomes.

During this stage, testing takes place in three levels of specificity (from smallest to largest):

1. Unit testing
2. Component testing
3. System testing

Unit Testing

A unit is the smallest piece of code that can be tested, usually methods or object classes. In unit testing, this small piece of code is tested explicitly by calling these functions with different input parameters. This is to see if the unit can handle all possible operations, attributes and state changes.

As I'm sure you can guess, having a test for each unit of code means you will have a lot of tests to run! Especially because you want to be able to run all your tests every time you make a change in the code. That is why it is important to automate these

tests as much as you can. Fortunately, test automation frameworks like JUnit make this simple: they have generic test classes that you can adapt to your specific test case. With these frameworks, it is possible to run all your tests and receive feedback in just a few seconds.

If an object you are testing depends on something to run that could slow down your testing process, you may want to create a mock object. For example, if your object fetches information from a huge database every time it runs, you could create a smaller database that the object runs through during tests. It is testing the same interface and functionality, but it will save you a lot of time.

Component Testing

Once the individual units have been tested, some units can be put together to create components. These components can then be tested to see how they interact with one another. We use a defined component interface to see how all the units function together, and so in component testing, the aim is to test this interface — to check that it is reacting as it should.

System Testing

Finally, we put all the components together to create the system. As you can imagine, there are only some things we'll be able to test when we get to this stage because it requires different components interacting with one another. If you recall Reuse-Oriented Engineering from *Agile Development*, this is the stage in which the reused components are integrated with the newly created ones.

During this stage, it should be tested that the system is ONLY achieving its specified requirements. We don't want any unintended 'features'. It is possible to automate this stage of testing. However, this depends on the predicted output. If the predicted output is particularly complex, if the quantity of output is huge, or if it is difficult to predict what the intended data should be, it may be better to test the system manually.

RELEASE TESTING

Once the system has been tested by the Developers, it is tested by an independent testing team just before the release of the product. Its intention is to make sure that the system meets the requirements of the clients and the users (validation testing), compared to system testing, which pushes the system to find potential

faults (defect testing). At this point in the process, testing is done to make sure the system doesn't fail during normal use.

Release testing is based on its own three approaches:

1. Requirements-based testing
2. Scenario testing
3. Performance testing

These are discussed below:

Requirements-Based Testing

In requirements-based testing, as the name suggests, a set of tests is created for each requirement of the system. During this process, it is important to keep records of the testing and the outcomes of each for traceability.

Scenario Testing

This approach had a more realistic nature. It tests several requirements at one time by testing the system in a scenario when it would typically be used.

Performance Testing

The final approach in release testing is performance testing. Here, everything is integrated and the system is put under stress tests to assess its performance and reliability. This approach reveals the system's failure behaviour and highlights any defects that arise when the load is pushed too far. This ensures that the system can accept the intended load, and system failures can be avoided.

USER TESTING

Once the client is happy with the system, it is then tested by the users themselves in the environment in which they would utilise the system. This may help surface issues that were not present in the controlled environment of the development space.

The three steps of user testing are:

1. Alpha testing
2. Beta testing

3. Acceptance testing

Alpha testing: In this first step, the system is tested by the user with the development team to see if further functionality needs to be added, or if unforeseen problems arise.

Beta testing: Once the system has passed alpha testing, the users then experiment with the system and raise further problems found.

Acceptance Testing: Finally, the users test the system themselves with real data to see if everything runs smoothly. If not, or if some requirements are not met, more development may be needed.

Compulsory Task

Follow these steps:

- Create a new text file called **testing.txt** inside this folder.
- Inside **testing.txt**, answer the following questions:
    - Why is it not necessary for a program to be completely free of defects before it is delivered to its customers? Explain your answer. Why can testing only detect the presence of errors, not their absence?
    - Some people argue that developers should not be involved in testing their own code and instead, all testing should be the responsibility of a separate team. What are the pros and cons of testing being carried out by the developers themselves?
    - A common approach to system testing is to test the system until the testing budget is exhausted and then deliver the system to customers. Discuss the ethics of this approach to system testing.

Optional Bonus Task

Follow these steps:

- Create a new program called **myUnitTest.java** inside this folder. Inside **myUnitTest.java**:
- Create a method that accepts the year that the user was born as a parameter and calculates the user's age. Assume that the age is calculated using the formula *age = currentYear - yearBorn*. The method should either return an integer value that indicates how old the user is or an appropriate error message if the user has entered an invalid year of birth.
- Read the appropriate documentation for creating a test using JUnit:
    - If you're using Eclipse: **https://www.eclipse.org/eclipse/news/4.7.1a/#junit-5-support**
    - If you're using IntelliJ IDEA: **https://blog.jetbrains.com/idea/2016/08/using-junit-5-in-intellij-idea/**
    - If you're using another development environment: **https://junit.org/junit5/docs/current/user-guide/#running-tests-ide**
- See how to create a test using JUnit here: **https://junit.org/junit5/docs/current/user-guide/#writing-tests**
- Write a test for the method that you have created.

Rate us
Share your thoughts

HyperionDev strives to provide internationally-excellent course content that helps you achieve your learning outcomes.

Think that the content of this task, or this course as a whole, can be improved or think we've done a good job?

Click here to share your thoughts anonymously.

References:

Sommerville, I. (2011). Software Engineering 9. 9th ed. Boston: Pearson Education, Inc., pp. 205-233.