Ahmed Almutawa & Parvinder Singh
HW3
CSCI3308
Boese

Test code: http://54.153.4.164/kata/edit/1EC734?avatar=buffalo
ID: 1EC&34
Avatar: Buffalo

Rather than test the entire codebase, we can utilize various testing methods to test parts of the code base individually to weed out the minor errors in each module; **unit testing** is a software **testing** method by which individual units of source code, sets of one or more computer program modules together with associated control data, usage procedures, and operating procedures, are tested to determine whether they are fit for use. By segmenting the code to manageable sizes to test, we can sure consistency throughout the codebase.

On the other hand, **Integration testing** is the phase in software **testing** in which individual software modules are combined and **tested** as a group. It occurs after unit **testing** and before validation **testing**. This is the logical follow-up to unit testing, where we combine the various units to perform a certain task and see if they can work together without error.

Following this, we have **Functional testing** which is a quality assurance (QA) process and a type of black box **testing** that bases its **test** cases on the specifications of the software component under **test**. This tests the overall behaviour and performance of the program, regardless of specific units and segments. While the previous testing methods work on a production level, functional testing is required as the client won't know what units failed as opposed to the entire code failing.

Lastly, **acceptance testing** is a **test** conducted to determine if the requirements of a specification or a contract are met. This interlinks the specifications and requirements of a software project set with the client with the produced code and project to ensure that the code performs the required jobs.

The advantages of having a largely tested code base is a more robust user experience and it acts as a preventative security measure. By testing the code, we expose flaws and bugs in the code that: 1) damage the user experience, 2) catch adhoc cases and 3) eliminate points of attack in the code by identifying where tests fail (e.g. buffer overflow exploits etc). By ensuring our inputs have plausible results and that we can account for the limit and special cases, we improve the UI and limit further maintenance and revision expenses. By also testing for functionality and acceptance, we also ensure the customer satisfaction of the clientele.