[Handout for L10P1]
Shipping with Confidence: Product-Level Quality Assurance

Product testing

Quality assurance (QA) is the process of ensuring that the software we build has the required quality levels.

Quality Assurance = Validation + Verification

QA involves checking of two aspects:

a) Have we built the right system? i.e., Are the requirements correct? This is called *validation*.
b) Have we built the system right? i.e., Have we implemented the requirements correctly? This is called *verification*.

Whether something belongs under validation or verification is not that important. What is more important is we do both, instead of limiting to verification (i.e., remember that the requirements can be wrong too).

Testing is the most common way of assuring quality, however there are other complementary techniques such as formal verification (explained in a previous handout). In this handout, we focus on testing the *product as a whole,* as opposed to developer testing that is performed on a partial system.

System testing

When we take a whole system instead of a part of the system and test it against the system specification, it is called *system testing*. System testing is typically done by a testing team (also called a QA team). We base the system test cases exclusively on the specified external behavior of the system. Sometimes, system tests go beyond the bounds defined in the specification. This is useful if we want to test whether the system fails 'gracefully' when pushed beyond its limits. For example, let us assume the software under test (SUT) is a browser and it should be able to handle web pages containing up to 5000 characters. One of our test cases can try to load a web page containing more than 5000 characters. The expected 'graceful' behavior can be 'abort the loading of the page and show a meaningful error message'. We can consider this test case as failed if the browser attempted to load the large file anyway and crashed.

Note that system testing includes testing against non-functional requirements too. Here are some examples:

- *Performance testing* - to ensure the system responds quickly.
- *Load testing* (also called *stress testing* or *scalability testing*) - to ensure the system can work under heavy load.
- *Security testing* – to test how secure is the system.
- *Compatibility testing*, *interoperability testing* – to check whether the system can work with other systems.
- *Usability testing* – to test how easy it is to use the system.
- *Portability testing* – to test whether the system works on different platforms.

Acceptance testing

Acceptance testing, also called User Acceptance Testing (UAT), is a validation type testing carried out to show that the delivered system meets the requirements of the customer. This too,

similar to the System testing, uses the whole system, but the testing is done against the requirements specification, not against the system specification. Note the two specifications need not be necessarily same. For example, requirements specification could be limited to how the system behaves in normal working conditions while the system specification can also include details on how it will fail gracefully when pushed beyond limits, how to recover, additional APIs not available for users (for the use of developers/testers), etc.

Acceptance testing comes after system testing. It is usually done by a team that represents the customer, and it is usually done on the deployment site or on a close simulation of the deployment site. UAT test cases are often defined at the beginning of the project, usually based on the use case specification. Successful completion of UAT is often a prerequisite to the project signoff.

Acceptance tests gives an assurance to the customer that system does what it is intended to do. Besides, acceptance testing is important because a system could work perfectly on the development environment, but fail in the deployment environment due to subtle differences between the two.

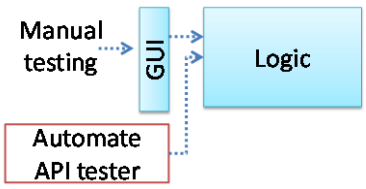
Alpha and Beta testing

Alpha testing is performed by the users, under a controlled condition set by the software development team. *Beta testing* is performed by a selected subset of target users of the system in their natural work setting. An *open beta release* is the release of not-yet-production-quality-but-almost-there software to the general population.

GUI testing

If a software product has a GUI, all product-level testing (i.e. the types of testing mentioned above) need to be done using the GUI. However, GUI testing is much harder than CLI (command line interface) testing or API testing, for the following reasons:

- Most GUIs contain a large number of different operations, many of which can be performed in any order.
- GUI operations are more difficult to automate that API testing. Reliably automating GUI operations and automatically verifying whether the GUI behaved as expected is harder than calling an operation and comparing its return value with an expected value. Therefore, automated regression testing of GUIs is rather difficult. However, there are testing tools that can automate GUI testing. For example, Selenium (http://seleniumhq.org/) can be used to automate testing of Web application UIs while certain editions for VisualStudio supports 'record replay' type GUI test automation.
- The appearance of a GUI (and sometimes even behaviour) can be different across platform and even based on environment. For example, a GUI can behave differently based on whether it is minimized or maximized, in focus or out of focus, and in a high resolution display or a low resolution display.



One approach to overcome the challenge of testing GUIs is to minimize logic aspects in the GUI. Then, we bypass the GUI to test the rest of the system using automated API testing. While this still requires the GUI to be tested manually, we can reduce the number of such manual test cases because most of the system has already been tested using automated API testing.

---End of Document---