Software Testing #3

What are the levels of testing?

What are the levels of testing?

- A level of software testing is a process where every unit or component of a software/system is tested
- These testing levels are designed to recognize missing areas and reconciliation between the development lifecycle states.
- There are mainly four testing levels are:
  - Unit Testing
  - Integration Testing
  - System Testing
  - Acceptance Testing

What are the levels of testing?

- A level of software testing is a process where every unit or component of a software/system is tested
- These testing levels are designed to recognize missing areas and reconciliation between the development lifecycle states.
- There are mainly four testing levels are:
  - Unit Testing
  - Integration Testing
  - System Testing
  - Acceptance Testing

Unit Testing

What is Unit Testing?

- UNIT Testing is defined as a type of software testing where individual units/ components of a software are tested
- Is done during the development (coding) of an application
- Also called **component testing**
- The objective of Unit Testing is to isolate a section of code and verify its correctness
- Usually performed by the developer
- Unit testing is first level of testing done before integration testing
- Due to time crunch or reluctance of developers to tests, QA engineers also do unit testing

Why Unit Testing done?

- Software developers attempt to save time by doing minimal unit testing
- Skipping on unit testing leads to higher Defect fixing costs during System Testing, Integration Testing
- Proper unit testing done during the development stage saves both time and money in the end.

Key reasons to do Unit Testing!

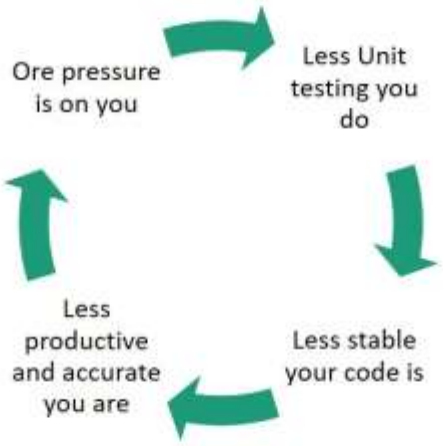
- Unit Tests fix bug early in development cycle and save costs.
- It helps understand the developers the code base and enable them to make changes quickly
- Good unit tests serve as project documentation
- Unit tests help with code re-use. Migrate both your code and your tests to your new project. Tweak the code till the tests run again.

How to do Unit Testing?

- Unit Testing is of two types-
  - Manual –
    - A manual approach to unit testing may employ a step-by-step instructional document.
  - Automated
    - A developer writes a section of code in the application just to test the function. They would later comment out and finally remove the test code when the application is deployed.
    - A developer could also isolate the function to test it more rigorously
    - Isolating the code helps in revealing unnecessary dependencies between the code being tested and other units or data spaces in the product

Unit Testing Myth

- **Myth:**
  - It requires time, and I am always overscheduled and don't have time
  - My code is rock solid! I do not need unit tests.
- **Truth** is Unit testing increase the speed of development.
- Coders/ developers think that Integration Testing will catch all errors and do not execute the unit test.
- Very simple errors which could have very easily found and fixed in unit tested take a very long time to be traced and fixed.



Unit Testing Advantage

- Due to the modular nature of the unit testing, we can test parts of the project without waiting for others to be completed
- Developers/Coders looking to learn what functionality is provided by a unit and how to use it can look at the unit tests to gain a basic understanding of the unit

Unit Testing Disadvantages

- Unit testing can't be expected to catch every error in a program. It is not possible to evaluate all execution paths
- Unit testing by its very nature focuses on a unit of code. Hence it can't catch integration errors or broad system level errors.

Unit Testing Best Practices

- Unit Test cases should be independent. In case of any enhancements or change in requirements, unit test cases should not be affected.
- Test only one code at a time.
- Bugs identified during unit testing are to be fixed before proceeding to the next phase in SDLC

Integration Testing

What is Integration Testing?

- Defined as a type of testing where software modules are integrated logically and tested as a group
- A typical software project consists of multiple software modules, coded by different programmers.
  - Integration Testing focuses on checking data communication amongst these modules.
- Other names-
  - 'I & T' (Integration and Testing)
  - String Testing
  - Thread Testing

Why do Integration Testing?

- Even we have tested software modules in unit tested, defects still exist for various reasons-
  - A Module, is designed by an individual software developer whose understanding and programming logic may differ from other programmers
  - Integration Testing becomes necessary to verify the software modules work in unity
  - Interfaces of the software modules with the database could be erroneous
  - External Hardware interfaces, if any, could be problem
  - Inadequate exception handling could cause issues.
  - there are wide chances of change in requirements by the clients. These new requirements may not be unit tested and hence system integration Testing becomes necessary.

Example of Integration Testing

Note-

- **Focuses mainly on the interfaces & flow of data/information between the modules and not on "How units are functioning?"**

Approaches of Integration Testing

Approaches of Integration Testing

Big Bang Approach

Incremental Approach

- Top Down Approach
- Bottom Up Approach
- Sandwich Approach - Combination of Top Down and Bottom Up

Big Bang Approach

- All component are integrated together at once and then tested
- Pros-
  - Convenient for small systems
- Cons-
  - Fault Localization is difficult.
  - Given the sheer number of interfaces that need to be tested in this approach, some interfaces link to be tested could be missed easily.
  - Since the Integration testing can commence only after "all" the modules are designed, the testing team will have less time for execution in the testing phase.
  - Since all modules are tested at once, high-risk critical modules are not isolated and tested on priority. Peripheral modules which deal with user interfaces are also not isolated and tested on priority

Incremental Approach

- Testing is done by joining two or more modules that are logically related
- And then the other related modules are added and tested for the proper functioning
- The process continues until all of the modules are joined and tested successfully.

What is Stub and Driver?

- Incremental Approach is carried out by using dummy programs called Stubs and Drivers.
- Stubs and Drivers do not implement the entire programming logic of the software module but just simulate data communication(or integration) with the calling module
- **Stub**: Is called by the Module under Test
- **Driver**: Calls the Module to be tested

Bottom-up Integration

- Each module at lower levels is tested with higher modules until all modules are tested.
- It takes help of Drivers for testing

Advantages

- Fault localization is easier.
- No time is wasted waiting for all modules to be developed unlike Big-bang approach

Disadvantages

- Critical modules (at the top level of software architecture) which control the flow of application are tested last and may be prone to defects.

Top-down Integration

- In Top to down approach, testing takes place from top to down following the control flow of the software system.
- Takes help of stubs for testing.

Advantages

- Fault Localization is easier.
- Possibility to obtain an early prototype.
- Critical Modules are tested on priority; major design flaws could be found and fixed first.

Disadvantages

- Needs many Stubs.
- Modules at a lower level are tested inadequately.

Hybrid/ Sandwich Integration

- In the sandwich/hybrid strategy is a combination of Top Down and Bottom up approaches.
- Here, top modules are tested with lower modules at the same time lower modules are integrated with top modules and tested.
- This strategy makes use of stubs as well as drivers.

How to do Integration Testing?

1. Prepare the Integration Tests Plan
2. Design the Test Scenarios, Cases, and Scripts.
3. Executing the test Cases followed by reporting the defects.
4. Tracking & re-testing the defects.
5. Steps 3 and 4 are repeated until the completion of Integration is successful.

Brief Description of Integration Test Plans

It includes the following attributes:

- Methods/Approaches to testing (as discussed above).
- Scopes and Out of Scopes Items of Integration Testing.
- Roles and Responsibilities.
- Pre-requisites for Integration testing.
- Testing environment.
- Risk and Mitigation Plans.

Entry and Exit Criteria of Integration Testing

Entry Criteria:

- Unit Tested Components/Modules
- All High prioritized bugs fixed and closed
- All Modules to be code completed and integrated successfully.
- Integration tests Plan, test case, scenarios to be signed off and documented.
- Required test environment to be set up for Integration testing

Exit Criteria:

- Successful Testing of Integrated Application.
- Executed Test Cases are documented
- All High prioritized bugs fixed and closed
- Technical documents to be submitted followed by release Notes.

Best Practices/ Guidelines for Integration Testing

- First, determine the Integration Test Strategy that could be adopted and later prepare the test cases and test data accordingly.
- Study the Architecture design of the Application and identify the Critical Modules. These need to be tested on priority.
- Obtain the interface designs from the Architectural team and create test cases to verify all of the interfaces in detail. Interface to database/external hardware/software application must be tested in detail.
- After the test cases, it's the test data which plays the critical role.
- Always have the mock data prepared, prior to executing. Do not select test data while executing the test cases

Thank You