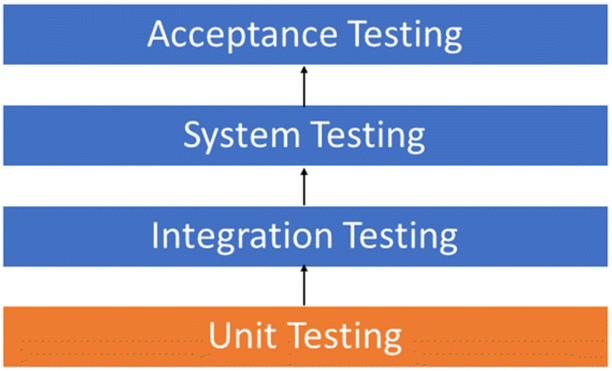
Unit Test and Integration Testing and interview questions

Wednesday, January 27, 2021 6:40 PM

Unit Testing

- UNIT TESTING is a type of software testing where individual units or components of a software are tested. The purpose is to validate that each unit of the software code performs as expected.
- Unit Testing is done during the development (coding phase) of an application by the developers
- A unit may be an individual function, method, procedure, module, or object
- 

- Unit Tests isolate a section of code and verify its correctness. Isolating the code helps in revealing unnecessary dependencies between the code being tested and other units or data space.
- Unit Testing is of two types
  - Manual
  - Automated
- Mock Objects: Unit testing relies on mock objects being created to test sections of code that are not yet part of a complete application. Mock objects fill in for the missing parts of the program. For example, you might have a function that needs variables or objects that are not created yet. In unit testing, those will be accounted for in the form of mock objects created solely for the purpose of the unit testing done on that section of code.
- **Unit Testing Tools**
  a. Junit: Junit is a free to use testing tool used for Java programming language. It provides assertions to identify test method. This tool test data first and then inserted in the piece of code.
  b. NUnit: NUnit is widely used unit-testing framework use for all .net languages. It is an open source tool which allows writing scripts manually. It supports data-driven tests which can run in parallel.
- Code coverage techniques used in Unit Testing are listed below:
  - Statement Coverage
  - Decision Coverage
  - Branch Coverage
  - Condition Coverage
  - Finite State Machine Coverage
- Techniques Within Unit Testing
  a. White box testing:In white-box testing, the tester knows the internal structure of the software including the code and can test it against the design and the requirements. Hence white box testing is also known as **transparent testing**.
  b. Black box testing:In black-box testing, the tester does not know the internal structures either the code of the software**.**
  c. Grey box testing: :This is also referred to as semi-transparent technique testing which means, the testers are only partially aware of the internal structure, functions, and designs along with the requirements
- Benefits Of Unit Testing
  - **Code quality improves**
  - **Detects bugs early**
  - **Easier changes and simplified integrations**
  - **Easy debugging process:**
- Best Practice
- **Code should be strong**
- **Understandable and reasonable**
- **Should be the single case:** Tests that defines multiple cases in one, are complex to work with. Thus writing a single case code is best practice, which makes the code easier to understand and debug

- **Allow automated tests:** The developers should make sure that the test runs in an automated form. It should be in a continuous delivery process or integration process

What is Integration Testing?

- **INTEGRATION TESTING** is 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. The purpose of this level of testing is to expose defects in the interaction between these software modules when they are integrated

- Integration Testing focuses on checking data communication amongst these modules. Hence it is also termed as **'I & T'** (Integration and Testing), **'String Testing'** and sometimes **'Thread Testing'**.

- Although each software module is unit tested, defects still exist for various reasons like

    - A Module, in general, 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
    - At the time of module development, 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

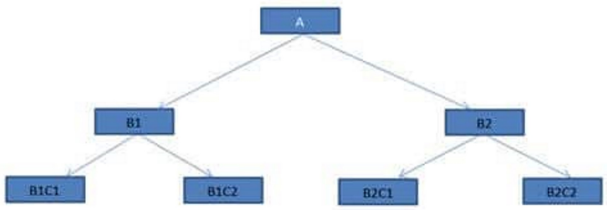
- **Example of Integration Test Case**
    - Sample Integration Test Cases for the following scenario: Application has 3 modules say 'Login Page', 'Mailbox' and 'Delete emails' and each of them is integrated logically. Here do not concentrate much on the Login Page testing as it's already been done in Unit Testing. But check how it's linked to the Mail Box Page.

Test Case ID	Test Case Objective	Test Case Description	Expected Result
1	Check the interface link between the Login and Mailbox module	Enter login credentials and click on the Login button	To be directed to the Mail Box
2	Check the interface link between the Mailbox and Delete Mails Module	From Mailbox select the email and click a delete button	Selected email should appear in the Deleted/Trash folder

Approaches, Strategies, Methodologies 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 Testing** is an Integration testing approach in which all the components or modules are integrated together at once and then tested as a unit. This combined set of components is considered as an entity while testing. If all of the components in the unit are not completed, the integration process will not execute.
Advantages:
    - Convenient for small systems.
Disadvantages:
    - 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.

- In the **Incremental Testing** approach, testing is done by integrating two or more modules that are logically related to each other and then tested for proper functioning of the application. Then the other related modules are integrated incrementally and the process continues until all the logically related modules are integrated and tested successfully
- Bottom-up testing, as the name suggests starts from the lowest or the innermost unit of the application, and gradually moves up. The Integration testing starts from the lowest module and gradually progresses towards the upper modules of the application. This integration continues till all the modules are integrated and the entire application is tested as a single unit.
- 

- In this case, modules B1C1, B1C2 & B2C1, B2C2 are the lowest module which is unit tested. Module B1 & B2 are not yet developed. The functionality of Module B1 and B2 is that it calls the modules B1C1, B1C2 & B2C1, B2C2. Since B1 and B2 are not yet developed, we would need some program or a "stimulator" which will call the B1C1, B1C2 & B2C1, B2C2 modules. These stimulator programs are called **DRIVERS**
- In simple words, **DRIVERS** are the dummy programs which are used to call the functions of the lowest module in a case when the calling function does not exist
- Top-down approach

    - This technique starts from the topmost module and gradually progress towards the lower modules. Only the top module is unit tested in isolation. After this, the lower modules are integrated one by one. The process is repeated until all the modules are integrated and tested.

    - In the context of our figure, testing starts from Module A, and lower modules B1 and B2 are integrated one by one. Now here the lower

modules B1 and B2 are not actually available for integration. So in order to test the topmost modules A, we develop "**STUBS**".

- "Stubs" can be referred to as code a snippet which accepts the inputs/requests from the top module and returns the results/ response. This way, in spite of the lower modules, do not exist, we are able to test the top module.

Stubs	**Driver**
Used in Top down approach	Used in Bottom up approach
Top most module is tested first	Lowest modules are tested first.
Stimulates the lower level of components	Stimulates the higher level of components
Dummy program of lower level components	Dummy program for Higher level component

Entry and Exit Criteria of Integration Testing

- Entry and Exit Criteria to Integration testing phase in any software development model
- **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.

Interview Questions in this Video : Basic introduction: - Tell me about yourself? - Tell me your day to day activity? - have you created framework from yourself? Java: - can you rate yourself in java/testNg/Selenium/Cucumber out of 1 to 10? - can you tell me OOPS concept and relate it with your framework? - What is the rule for overriding? - How to achieve 100% abstraction in Java? - What is Abstract class and how it is different with Interface? - What do you mean by Static keyword in Java? - Can we overload main method? - Can we override main method? - What is Exception in java? - What is the difference between throw and throws keyword in java? Selenium: - Which locator you use mostly and why? - What is the difference between findElement and findElements? - How to handle multiple windows in Selenium? - Difference between windowHandle and windowHandles? - How to navigate from 1 window to another in Selenium? - What is the difference between Implicit and Explicit wait? - Can we combine implicit and explicit wait? - Can you list some Exceptions in Selenium? - What is staleElement exception and how to handle? TestNg: - What is assert in testNg? - What is hard and soft assert and what is difference? - Can you list down some of the annotation of testNg and what is order of annotation? - Can we set priority to test cases in testNg and what if I add negative priority? Cucumber: - What all files required in Cucumber and can you please elaborate all? - What is the difference between scenario and scenario outline? - What is hooks in Cucumber? - Which one is better between Cucumber and testNg? Git: - What is the use of Git? - can you list some Git commands which you use in daily? - How to handle merge conflict? Jenkins: - How to create jobs in Jenkins? Manual Testing: - What is the difference between smoke and sanity testing? - What is defect life cycle? Agile: - What is Agile ceremonies? - What is the difference between epic, story and task?