Need for Testing **(Refer lecture handwritten notes.)***

goals of software testing

1. The first and the foremost objective of software testing is to ensure that it fulfills all the requirements of the customer, which means firstly understanding the customer requirements and then ensuring it behaves in the similar manner.
2. Second main objective is to find out the defects or issues occurring in the application before it are encountered by the end user.
3. Thirdly, to evaluate the overall performance of the application.
4. Providing a high quality product to the end user.
5. To evaluate security related issues for the application under test.
6. Creating a strong test scenario which in turn ensures that not a single scenario is missed.
7. Re-factoring the legacy test scenarios according to changed/updated functionality of the application as per the changing requirements of the end user.
8. Developing a test plan for the stakeholders which basically define the testing strategies, testing types to be performed during testing, browsers on which application will be tested etc.
9. Creating use cases can also be considered as major objective of software testing.
10. To ensure integration among various modules are working fine or if there are any third party integrations in the application then they should be working correctly even if they are not under the scope of the application.

Testing strategies **(Refer lecture handwritten notes for theory in details.)***

Strategy of testing

A strategy of software testing is shown in the context of spiral.

Following figure shows the testing strategy:

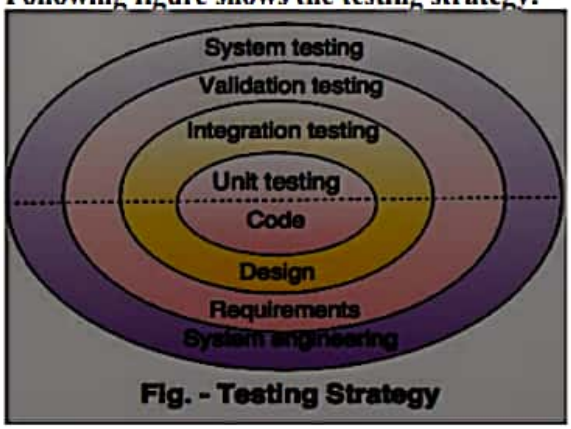


Fig. - Testing Strategy

These steps are shown in following figure:

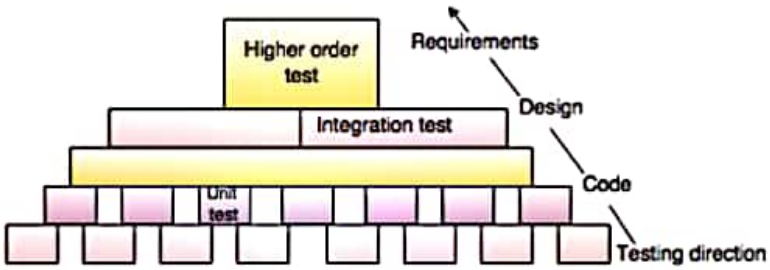


Fig.- Steps of software testing

Following are the issues considered to implement software testing strategies.

- Specify the requirement before testing starts in a quantifiable manner.
- According to the categories of the user generate profiles for each category of user.
- Produce a robust software and it's designed to test itself.
- Should use the Formal Technical Reviews (FTR) for the effective testing.
- To access the test strategy and test cases FTR should be conducted.
- To improve the quality level of testing generate test plans from the users feedback.

Following figure shows the unit testing:

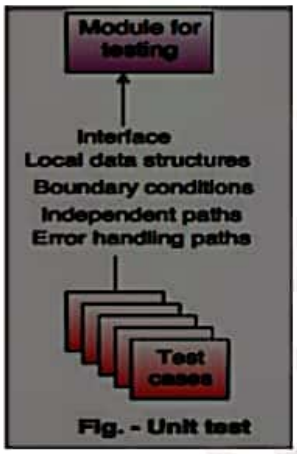


Fig. - Unit test

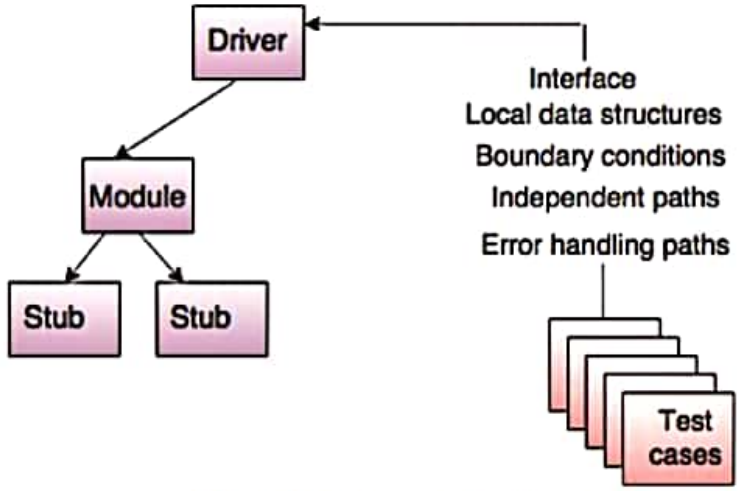


Fig. - Unit test environment

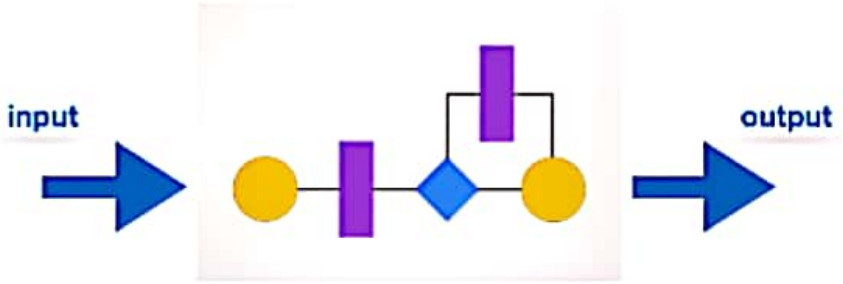
Black-box testing (Refer lecture handwritten notes for theory and sums in details.)*
It is carried out to test functionality of the program. It is also called 'Behavioral' testing. The tester in this case, has a set of input values and respective desired results. On providing input, if the output matches with the desired results, the program is tested 'ok', and problematic otherwise.

In this testing method, the design and structure of the code are not known to the tester, and testing engineers and end users conduct this test on the software.

Black-box testing techniques:

Equivalence class - The input is divided into similar classes. If one element of a class passes the test, it is assumed that all the class is passed.
Boundary values - The input is divided into higher and lower end values. If these values pass the test, it is assumed that all values in between may pass too.
Cause-effect graphing - In both previous methods, only one input value at a time is tested. Cause (input) – Effect (output) is a testing technique where combinations of input values are tested in a systematic way.

White-box testing (Refer lecture handwritten notes for other types and sums in details.)*
It is conducted to test program and its implementation, in order to improve code efficiency or structure. It is also known as 'Structural' testing.



In this testing method, the design and structure of the code are known to the tester. Programmers of the code conduct this test on the code.

The below are some White-box testing techniques:

Control-flow testing - The purpose of the control-flow testing to set up test cases which covers all statements and branch conditions. The branch conditions are tested for both being true and false, so that all statements can be covered.

Data-flow testing - This testing technique emphasis to cover all the data variables included in the program. It tests where the variables were declared and defined and where they were used or changed.

Acceptance Testing (Refer lecture handwritten notes for difference in details.)*

When the software is ready to hand over to the customer it has to go through last phase of testing where it is tested for user-interaction and response. This is important because even if the software matches all user requirements and if user does not like the way it appears or works, it may be rejected.

- **Alpha testing** - The team of developer themselves perform alpha testing by using the system as if it is being used in work environment. They try to find out how user would react to some action in software and how the system should respond to inputs.
- **Beta testing** - After the software is tested internally, it is handed over to the users to use it under their production environment only for testing purpose. This is not as yet the delivered product. Developers expect that users at this stage will bring minute problems, which were skipped to attend.

verification and validation and discuss its importance in project.

Verification	Validation
Are we building the system right?	Are we building the right system?
Verification is the process of evaluating products of a development phase to find out whether they meet the specified requirements.	Validation is the process of evaluating software at the end of the development process to determine whether software meets the customer expectations and requirements.
The objective of Verification is to make sure that the product being develop is as per the requirements and design specifications.	The objective of Validation is to make sure that the product actually meet up the user's requirements, and check whether the specifications were correct in the first place.
Following activities are involved in Verification: Reviews, Meetings and Inspections.	Following activities are involved in Validation: Testing like black box testing, white box testing, gray box testing etc.
Verification is carried out by QA team to check	Validation is carried out by testing team.

whether implementation software is as per specification document or not.	
Execution of code is not comes under Verification.	Execution of code is comes under Validation.
Verification process explains whether the outputs are according to inputs or not.	Validation process describes whether the software is accepted by the user or not.
Verification is carried out before the Validation.	Validation activity is carried out just after the Verification.
Following items are evaluated during Verification: Plans, Requirement Specifications, Design Specifications, Code, Test Cases etc,	Following item is evaluated during Validation: Actual product or Software under test.
Cost of errors caught in Verification is less than errors found in Validation.	Cost of errors caught in Validation is more than errors found in Verification.
It is basically manually checking the of documents and files like requirement specifications etc.	It is basically checking of developed program based on the requirement specifications documents & files.

Conclusion on difference of Verification and Validation in software testing:

1. Both Verification and Validation are essential and balancing to each other.
2. Different error filters are provided by each of them.
3. Both are used to finds a defect in different way, Verification is used to identify the errors in requirement specifications & validation is used to find the defects in the implemented Software application

McCall's Quality Factor **(Refer lecture handwritten notes.)**

Software Configuration Management:

Software configuration management is an important element of software quality assurance.
Its primary responsibility is the control of change. However, SCM is also responsible for the identification of individual SCIs and various versions of the software, the auditing of the software configuration to ensure that it has been properly developed, and the reporting of all changes applied to the configuration.

Any discussion of SCM Introduces a set of complex questions:

- How does an organization identify and manage the many existing versions of a program (and its documentation) in a manner that will enable change to be accommodated efficiently?
- How does an organization control changes before and after software is released to a customer?
- Who has responsibility for approving and ranking changes?
- How can we ensure that changes have been made properly?
- What mechanism is used to appraise others of changes that are made?

These questions lead us to the definition of five SCM tasks:

1. *identification,*
2. *version control,*
3. *change control,*
4. *configuration auditing,* and
5. *reporting.*