AAT-II

Explain about that

1. User interface design:-

Interface Design has three "golden rules".

1) Place the user in Control
2) Reduce the user's memory load.
3) Make the interface consistent.

These golden rules actually form the basis for a set of user interface design principles that guide this important software design action.

1) Place the user in Control:-

During a requirements-gathering session for a major new information system a key user was asked about the attributes of the windows-oriented graphical interface.

Most interface constraints and restrictions that are imposed by a designer are intended to simplify the mode of interaction.

In many cases, the designer might introduce constraints and limitations to simplify the implementation of the interface.

- Define interaction modes in a way that does not force a user into unnecessary (or) undesired actions:- An interaction mode is the current state of the interface.
- Provide for flexible interaction:- Because different interaction preferences, choices.

- Allow user interaction to be interruptible and undoable. Even when involved in a sequence of actions.
- Streamline interaction as skill levels advance and allow the interaction to be customized:- Users often find they perform the same sequence of interactions repeatedly.
- Hide technical internals from the casual user:- The user interface should the user into the virtual world of the application.
- Design for direct interaction with objects that appear on the screen: The user feels a sense of control when able to manipulate the objects.

2) Reduce the user's Memory Load:-

The more a user has to remember, the more error-prone interaction with the system will be. ~~It is for~~

- Reduce demand on short-term memory: when users are involved in complex tasks, the demand on short-term memory can be significant.
- Establish meaningful defaults:- The initial set of defaults should make sense for the average user
- Define shortcut that are intuitive:- when mnemonics are used to accomplish a system function.

- The visual layout of the interface should be based on a real world metaphor.
- Disclose information in a progressive fashion.

3) Make the Interface Consistent:-

The interface should present and acquire information in a consistent fashion.

This implies that:

1) all visual information is organized according to a design standard that is maintained throughout all screen display

2) Input mechanism are constrained to a limited throughout all screen display. Set that is used consistently throughout the application and.

3) Mechanisms for navigating from task to task are consistently defined and implemented.

Mandel defines a set of design principles that help make the interface consistent.

- Allow the user to put the current task into a meaningful context: Many interface implement complex layers of interactions with dozens of screen images. It is important to provide indicators.

- maintain consistency across a family of applications :- A set of applications should all implement the same design rules so that consistency is maintained for all interaction.
- If past interactive models have created user expectations, do not make changes unless there is a compelling reason to do so:- Once a particular interactive sequence has become a de facto standard, the user expects this in every application she encounters. A change alt-so to invoke scaling will cause confusion.

Interface Design Steps:-

Once interface analysis has been completed, all tasks required by the end-users have been identified in detail and the interface design activity commences. Interface design, like all software engineering design, is an iterative process. Each user interface design step occurs a number of times, each elaborating and refining information developed in the preceding step.

Although many different user interface design models have been proposed, all suggest some combination of the following steps:-

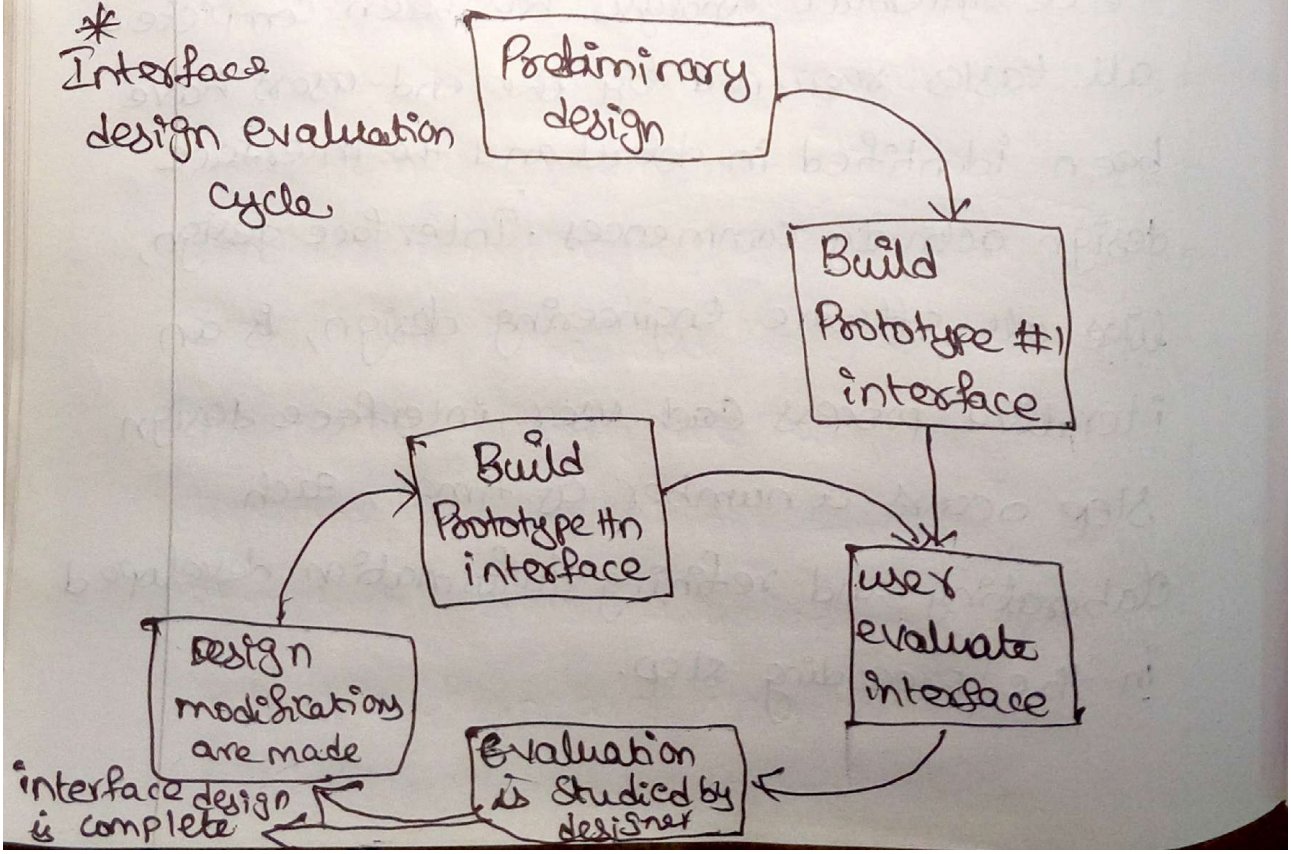
1) Using information developed during interface analysis define interface objects and actions.

2) Define events that will cause the state of the user interface to change. Model this behaviour.

3) Depict each interface state as it will actually look to the end-user.

4) Indicate how the user interprets the state of the system from information provided through the interface.

* Interface design evaluation cycle.



2) Explain in detail about Testing Tactics:-

Ans: Software Testing Fundamentals:-

Fundamental testing goals and principles were discussed. Recall that the goal of testing is to find errors and that a good test is one that has a high probability of finding an error. Therefore, a software engineer should design and implement a computer-based system (or) a product with "testability" in mind. At the same time, the tests themselves must exhibit a set of characteristics that achieve the goal of finding the most errors with a minimum of effort.

Testability:- James Bach provides the following definition for testability: "Software testability is simply how easily can be tested." The following characteristics lead to testable software.

Operability:- "The better it works, the more efficiently it can be tested." If a system is designed and implemented with quality in mind, relatively few bugs will block the execution of tests, allowing testing to progress without fits and starts.

Observability:- "what you see is what you test." Inputs provided as part of testing distinct outputs. System states and variables are visible or) queriable during execution. Incorrect output is easily identified. Internal errors are automatically detected and reported. Source code is accessible.

Controllability:- "The better we can control the software, the more the testing can be automated and optimized." Software and hardware states and variables can be controlled directly by the test engineer. Tests can be conveniently specified automated and reproduced.

Decomposability:- "By controlling the scope of testing we can more quickly isolate Problems and perform smarter retesting." The software system is built from independent modules that can be tested independently.

Simplicity:- "The less there is to the test the more quickly we can test it." The program should exhibit functional simplicity. structural simplicity and code simplicity. a coding standard is adopted for ease of inspection and maintenance.

stability:- "The fewer the changes, the fewer the disruptions to testing." changes to the software are infrequent, controlled when they do occur, and do not invalidate existing tests. The software recovers well from failures.

understandability:- "The more information we have the smarter we will test." the architectural design and the dependencies between internal, external and shared components are well understood. Technical documentation is instantly accessible, well organized, specific and detailed, and accurate. changes to the design are communicated to testers.

The attributes suggested by Bach can be used by a software engineer to develop a software configuration that is amenable to testing.

Test characteristics:- And what about the tests themselves? Kaner, Falk, and Nguyen suggest the following attributes of a "good" test.

1) A good test has a high probability of finding an error:- To achieve this goal the tester must understand the software and attempt to develop a mental picture.

2) A good test is not redundant:- Testing time and resources are limited. There is no point in conducting a test that has the same purpose as another test.

3) A good test should be "best for breed":- In a group of tests that have a similar intent, time and resource limitations may mitigate toward the execution of only a subset of these tests.

4) A good test should be neither too simple nor too complex:- Although it is sometimes possible to combine a series of tests into one test case.

White-Box Testing:-

White-Box Testing sometimes called glass-box Testing is a test case design philosophy.

1) guarantee that all independent paths with in a module have been exercised at least once.
2) Exercise all logical decision on their true and false sides,
3) Execute all loops at their boundaries and within their operation bounds and
4) Exercise internal data structures to ensure their validity.

Black-Box Testing:- Black-box testing also called behaviour testing, focuses on the functional requirements of the software. Black-box testing attempts to find errors in the following categories:- 1) Incorrect (or) missing functions.

2) Interface errors

3) errors in data structures (or) external database access

4) behaviour (or) performance errors

5) initialization and termination errors.

3) Explain in detail about testing strategies.

Ans:- Testing is a set of activities that can be planned in advance and conducted systematically For this reason a template for software testing a set of steps into which we can place specific test case design techniques and testing methods. A number of software testing strategies have been proposed in the literature. All provide the software developer with a template for testing and all have the following generic characteristics:-

- To perform effective testing a software team should conduct effective formal technical reviews. By doing this, many errors

will be eliminated before testing commences.

- Testing begins at the component level and works "outward" toward the integration of the entire computer-based system.
- Different testing techniques are appropriate at different points in time
- Testing is conducted by the developer of the software and an independent testing.
- Testing and debugging are different activities but debugging must be accomodate in any testing strategy.

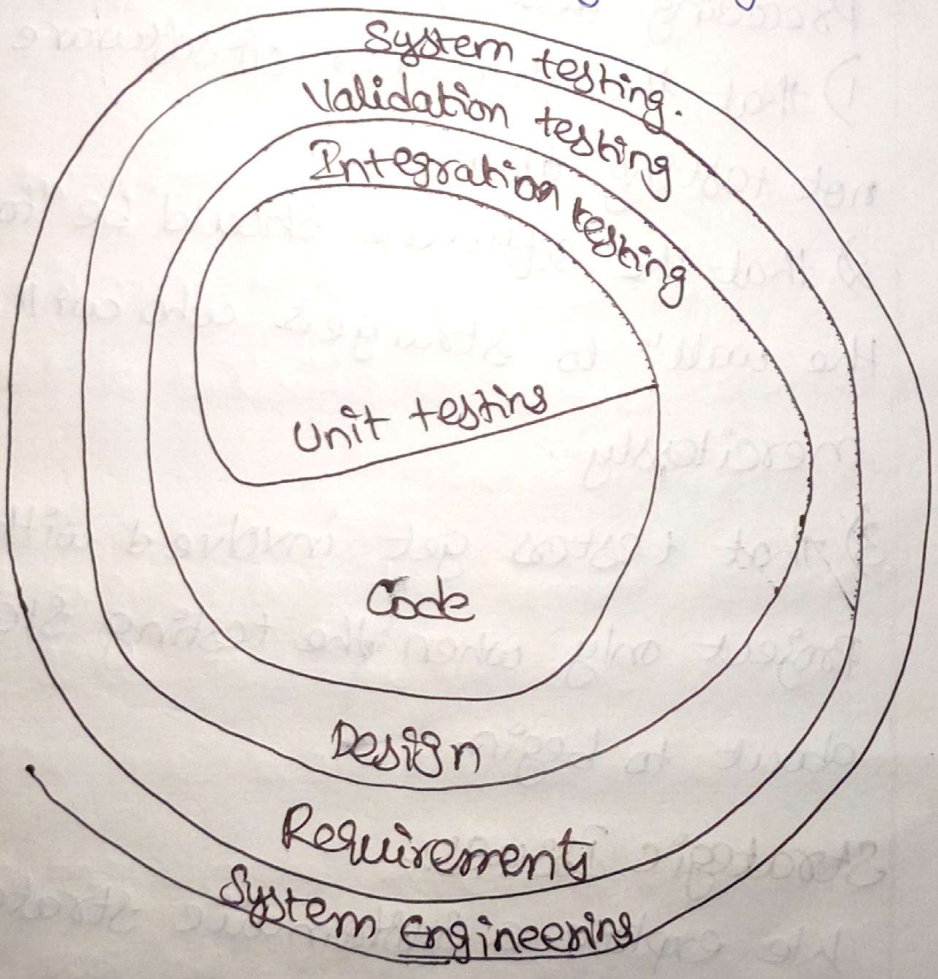
A Strategy for software testing must accommodate low-levels test that are necessary to verify that a small source code segment has been correctly implemented as well as higher-level tests that validate major system functions against customer requirements. A strategy must provide guidance for the practitioner and a set of milestones for the manager. Because the steps of the test strategy occur at a time when deadline pressure begins to rise, progress must be measurable

Verification and Validation:-

Verification refers to the set of activities that ensure that software correctly implements a specific function. Validation refers to a different set of activities that ensure that the software that has been built is traceable to customer requirements.

Verification:- Are we building the product right?

Validation:- Are we building the right product?



Testing Strategy

Organization for Software Testing:- For every software project, there is an inherent conflict of interest that occurs as testing begins. the people who have built the software are now asked to test the software.

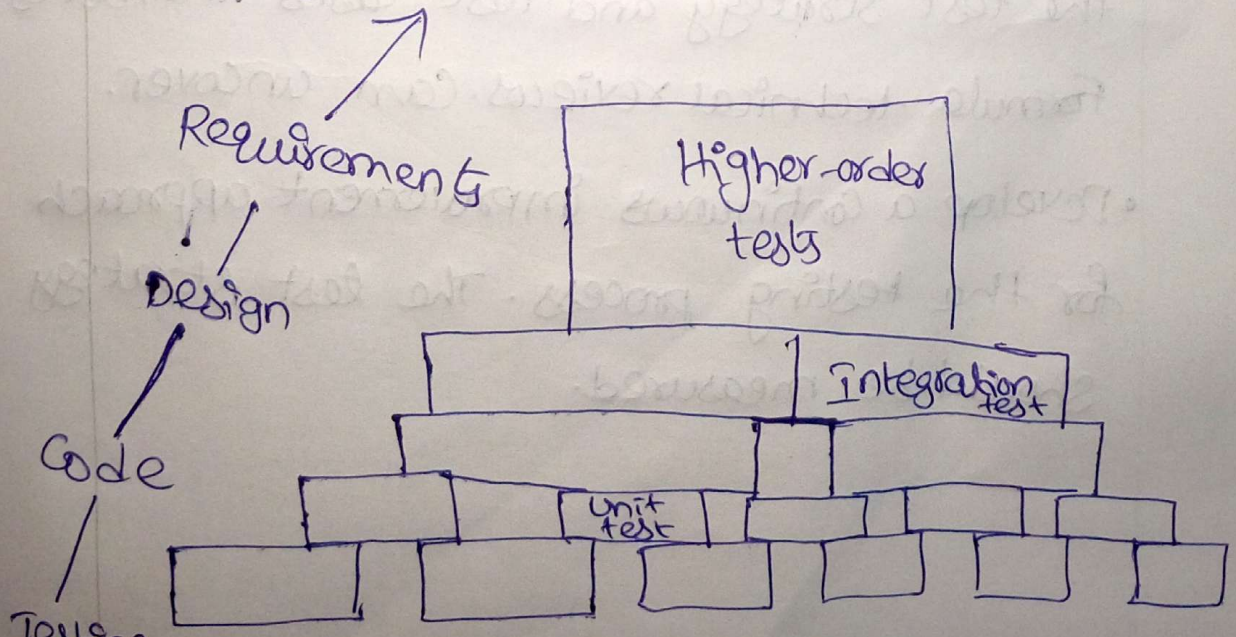
There are often a number of misconceptions that can be erroneously inferred from the preceding discuss

1) that the developer of software should do not testing at all.
2) that the software should be "tossed over the wall" to strangers. who will test it mercilessly.
3) that testers get involved with the project only when the testing steps are about to begin.

Strategic Issues:-

We explore a systematic strategy for software testing. But even the best strategy will fail if a series of overriding issues are not addressed. The following issues must be addressed if a successful software testing strategy is to be implemented

- Specify product requirements in a quantifiable manner long before testing commences :- Although the overriding objective of testing is to find errors, a good testing strategy also assess other quality characteristics such as portability, maintainability and usability.
- State testing objectives explicitly :- The specific objectives of testing should be stated in measurable terms.
- Understand the users of the software and develop a profile for each user category. Use-cases that describe the interaction scenario for each class of users can reduce overall testing effort by focusing testing on actual use of the product.



Software testing steps

- Develop a testing plan that emphasizes "rapid cycle testing":- learn to test in rapid cycles of customer-useful at least field "triable" increments of function and quality improvement.
- Build "robust" software that is designed to test itself:- Software should be designed in a manner that uses antibugging techniques.
- Use effective formal technical reviews as a filter prior to testing. Formula technical reviews can be as effective as testing in uncovering errors.
- Conduct formal technical reviews to assess the test strategy and test cases themselves Formula technical reviews can uncover.
- Develop a continuous improvement approach for the testing process. The test strategy should be measured.