Software Engineering

UNIT 5 (PART 1)

TESTING

Software Quality

*Definition : **Useful product** built through **effective software process**, wherein it adds **value to producers and consumers**.*

Effective software process results in good management, problem analysis and design.

Useful product is something that satisfies the user needs by giving desired functions and features.

Value is added to producer in the form of product revenue, less rework and reputation. Value for the consumer is in the form of profit in some business environment by implementing the product.

Different ways in which quality can be viewed :

Transcendental view : Quality can be recognized, not defined.

User view : Whether the product meets user's specific goals.

Manufacturer's view : Whether the product conforms to the original specifications.

Product view : Functions and features define them.

Value based view : Based on the money customer is willing to pay.

*Quality is defined by how well a product is designed and how well it conforms to the design, i.e, it depends on **Quality of design** & **Quality of conformance.***

Software Quality

Software quality product is defined in term of its fitness of purpose. That is, a quality product does precisely what the users want it to do. For software products, the fitness of use is generally explained in terms of satisfaction of the requirements laid down in the SRS document. Although "fitness of purpose" is a satisfactory interpretation of quality for many devices such as a car, a table fan, a grinding machine, etc.for software products, "fitness of purpose" is not a wholly satisfactory definition of quality.

Example: Consider a functionally correct software product. That is, it performs all tasks as specified in the SRS document. But, has an almost unusable user interface. Even though it may be functionally right, we cannot consider it to be a quality product.

The modern view of a quality associated with a software product has several quality factors such as the following:

Portability: A software device is said to be portable, if it can be freely made to work in various operating system environments, in multiple machines, with other software products, etc.
Usability: A software product has better usability if various categories of users can easily invoke the functions of the product.
Reusability: A software product has excellent reusability if different modules of the product can quickly be reused to develop new products.
Correctness: A software product is correct if various requirements as specified in the SRS document have been correctly implemented.
Maintainability: A software product is maintainable if bugs can be easily corrected as and when they show up, new tasks can be easily added to the product, and the functionalities of the product can be easily modified, etc.

Software Quality Management System

A quality management system is the principal methods used by organizations to provide that the products they develop have the desired quality.

A quality system subsists of the following:

Managerial Structure and Individual Responsibilities: A quality system is the responsibility of the organization as a whole. However, every organization has a sever quality department to perform various quality system activities. The quality system of an arrangement should have the support of the top management. Without help for the quality system at a high level in a company, some members of staff will take the quality system seriously.

Quality System Activities: The quality system activities encompass the following:

Auditing of projects

Review of the quality system

Development of standards, methods, and guidelines, etc.

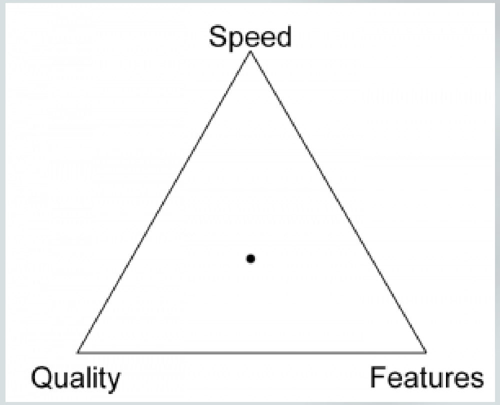
Production of documents for the top management summarizing the effectiveness of the quality system in the organization.

THE SOFTWARE QUALITY DILEMMA

The dilemma : Customers demand good quality software which must be delivered earlier as possible.

Hence arise a dilemma between quality, speed and features.

When you’re forced with quality dilemma try to achieve balance.



1) **Good enough software**

- Provides high quality functions user desires
- Delivers other specialized functions with known bugs
- Hoping users will overlook the bugs
- For some application domains, delivering as incremental versions helps getting into market fast
- Certain embedded systems can’t be made with known bugs and can have severe aftereffect

2) Cost of Quality

Quality has cost, but lack of quality costs user and producer in many ways

Cost of conformance –

- **Cost of prevention** - costs of : management activities needed to plan quality control activities, technical activities for designing a complete system, test planning, training
- **Cost of appraisal** – costs of : technical reviews of work products, data collection, testing

Cost of non conformance –

- **Internal failures** - errors found prior delivery : costs for rework, side effects while reworking, data collection
- **External failures** – errors found after product is delivered : costs of solving complaints, help line support, replacement

3) Risks

Apart from costs poor quality can be the result of unskilled labour as for design and coding and it may lead to severe accidents while embedded in health machines and other critical machines.

4) Negligence and liability

Developers negligible towards good software practices and users liable for giving stable requirements

5) Quality and security

Secure systems provide quality by not allowing unauthorized access, and should be taken care at the time of architectural design.

6) Impact of Management Actions

Estimation decisions, scheduling decisions, risk-oriented decisions all affect quality of a software

Achieving software quality

Software engineering methods

First step towards quality is understanding the problem at hand, then applying skilled analysis and design methods that pave way for high quality

Project management techniques

Management decisions for estimation of cost and time, scheduling, risk planning has greater impact on quality

Quality control actions

Quality control activities is a set of actions that ensure whether work products meet quality goals, this may include reviewing models, inspecting code, checking for errors in the way code is implemented

Software quality assurance

Includes a set of auditing and reporting functions used to analyse completeness of quality control actions, it also provides data related to product quality that can be utilized to improve applied methods

TESTING

- Executing a program for finding bugs.
- Process of verifying and validating whether the product meets the all type of requirements.

Test cases :

- Specifications of input to test and output expected.
- Behaviour of system in test cases determines its performance.

STRATEGIC APPROACH TO TESTING

Test strategy :

- Defines a planned and systematic testing approach for software development life cycle.
- Should contain test planning, test case design, test execution, data collection, evaluation.

DIFFERENT STRATEGIC APPROACHES

Analytical testing strategy – test conditions formulated after analysing test basics, example: risk based testing, requirement based testing

Model based testing strategy – creates model for existing or expected situation for the product

Methodical testing strategy – follows predefined quality standard like those specified by ISO

Standards or Process compliant testing strategy – follows guidelines specified by committee for standards

Reactive – based on defects found in system after delivery

Consultative – consultations from stakeholders considered for test conditions

A ***test strategy document*** *contains test levels, roles of stakeholders, testing tools, metrics, test plan, acceptance criteria and information on scheduling, standards, risk etc.*

GENERIC CHARACTERISTICS OF TESTING STRATEGIES

- Promote effective technical reviews
- Start testing at component level and move outwards
- Apply different techniques at different points of time according to appropriateness
- Conduct testing by collaboration of developer and an independent test group
- Testing and debugging must be considered different activities, but debugging should be included in testing.

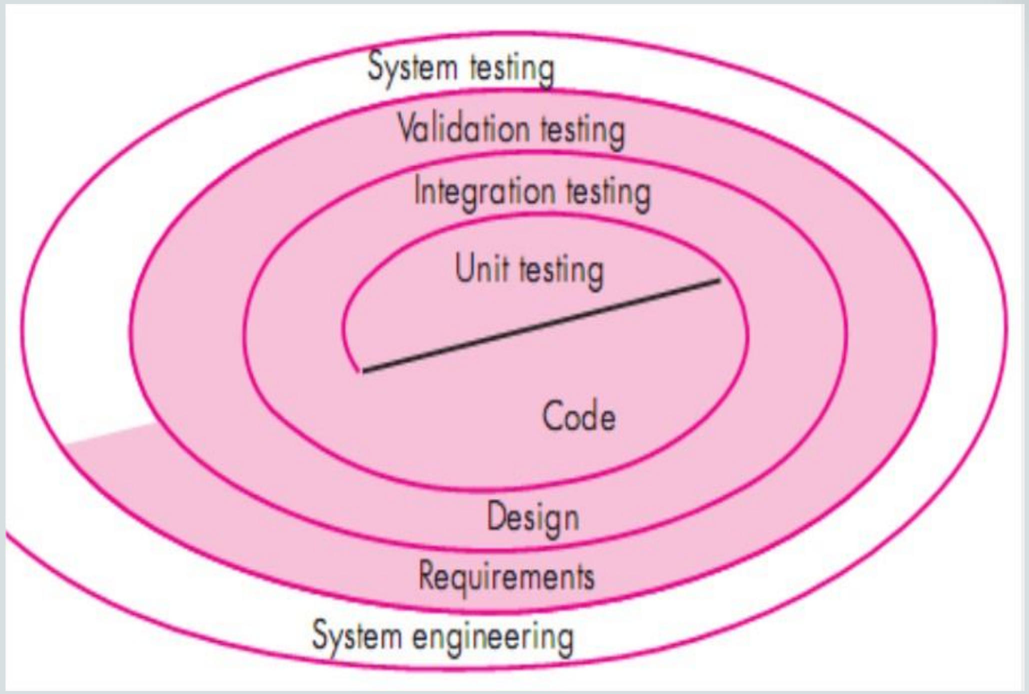
- Testing can be considered as an important part of **verification and validation (V&V)**, which includes technical reviews, performance monitoring, documentation review, qualification testing, installation testing and other such activities. Since quality starts building from the start of a software process, it should be tested throughout process and should not be moved to the end.

- While **organizing for software testing**, an independent test group(ITG) must be considered since testing by developer alone will be prone to the psychological fact that the person's intention will be to show the program works and not in unveiling errors. Hence, an ITG should be a part of project development team.

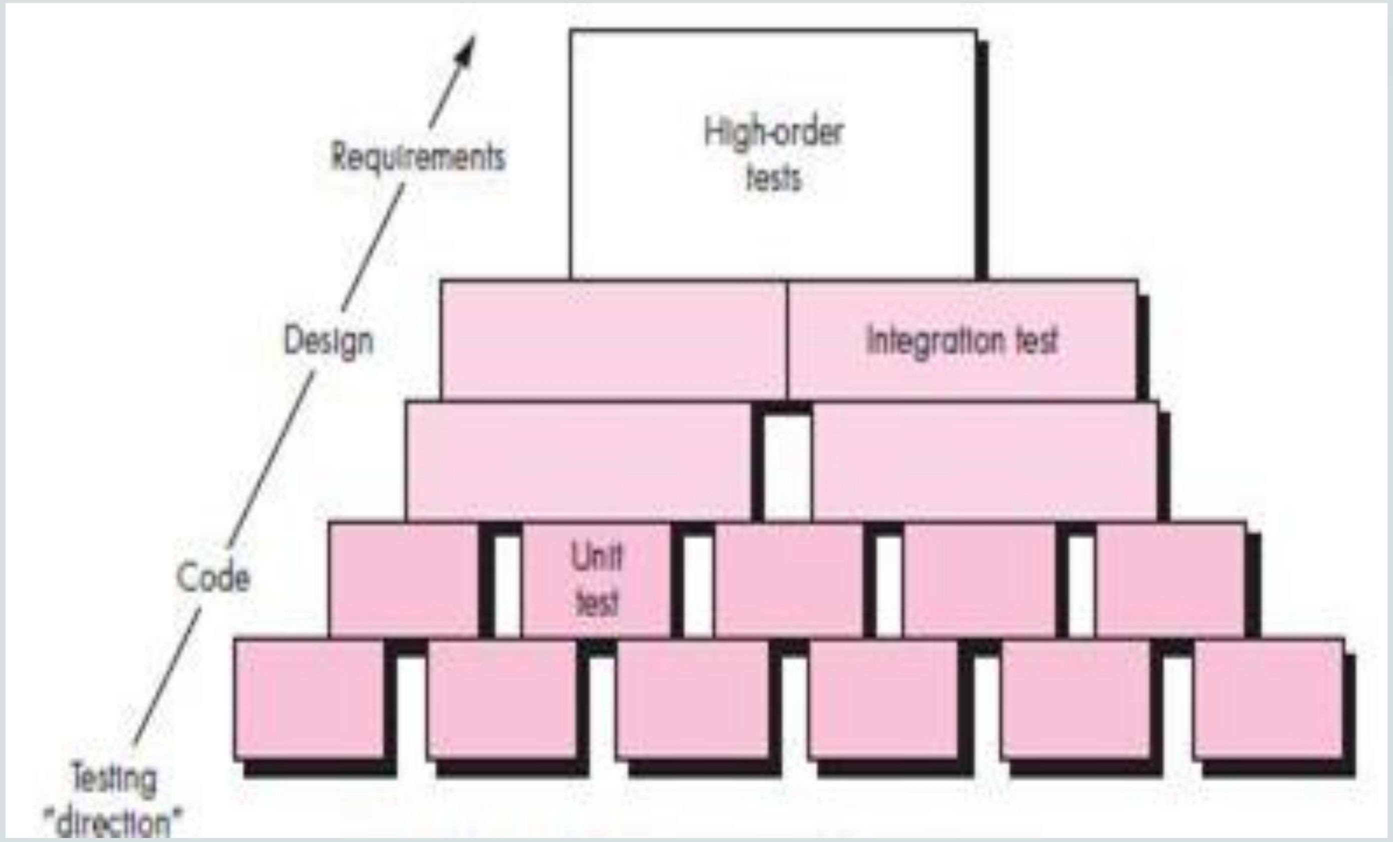
- **Completion of testing** is when the no critical defects exists, specified test cases are applied and resultant data is satisfying.

SOFTWARE TESTING STRATEGY

Based on **spiral structure**, software process spiral inwards beginning with defining role of software, moving inwards with requirement analysis, then designing and coding. For testing, it begins at the innermost point and spiral outwards. Starting at unit level, ***unit testing*** is done. ***Integration testing*** is done as we move outward based on the software architecture. ***Validation testing*** validates requirements against software. Finally, ***system testing*** is carried out where software and system elements are tested as a whole.



In a **procedural fashion**, testing occurs in a series of four steps which are similar to ones spiral model.
It includes unit testing, integration testing and higher order tests that include validation and system testing.



STRATEGIC ISSUES

The following points should be followed otherwise there is a chance for issues in any type of test strategy.

- Product requirements must be specified in a quantifiable manner before testing
- State test objectives explicitly which can be measured
- Understand users of product and create profile for each category
- Promote ***rapid cycle testing***
- Software should be able to test itself up to a level
- Promote technical reviews throughout software process and for test cases themselves
- The testing process should be improved continuously

Test Strategies For Conventional Software

- Unit Testing
- Integration Testing
- Regression Testing
- Smoke Testing
- Validation Testing
- System Testing
- Black Box Testing
- White Box Testing

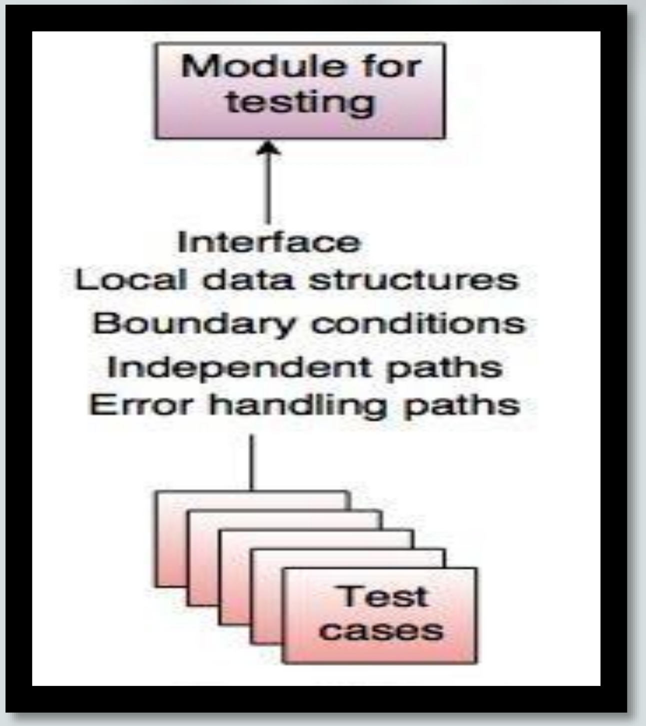
The testing strategy begins with testing of individual units, then by testing integration of these units and finally tests on the fully constructed system.

Unit Testing

- Focus on the smallest unit of software design, i.e module or software component.
- Based on component-level design the internal processing logic and data structures within components are tested.

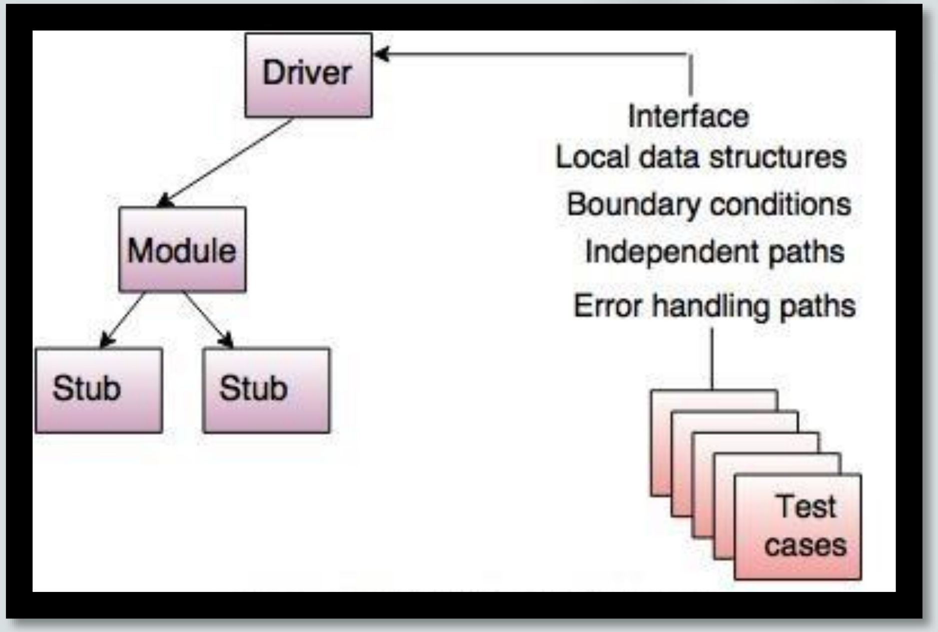
Unit Test Considerations

- Module interface tested to check proper data flow in and out.
- Tests whether local data structures maintain data integrity.
- All control paths of execution tested.
- Boundary conditions are checked.
- Error handling paths are executed and tested.



Unit Test Procedures

- Test cases and expected test results are outlined.
- **Drivers** and **stubs** are introduced for eligible units.
- Drivers act as main program, accepts test cases and passes to unit to be tested, also prints result.
- Stubs act as subordinate modules with which unit under test communicates.
- Components/units with high cohesion makes unit testing easier and lowers the overhead of creating sophisticated stubs.



Integration Testing

- Through integration testing we systematically build whole software architecture while testing to uncover ***interfacing*** errors.
- In ***big bang*** approach all components are combined and tested as whole. Here, error correction is difficult.
- In ***incremental integration*** testing is done in small increments after adding one or more components.
- Incremental integration can be ***top-down*** **or** ***bottom-up integration.***

Top-down integration

Beginning with main control moves downward by adding subordinate modules.

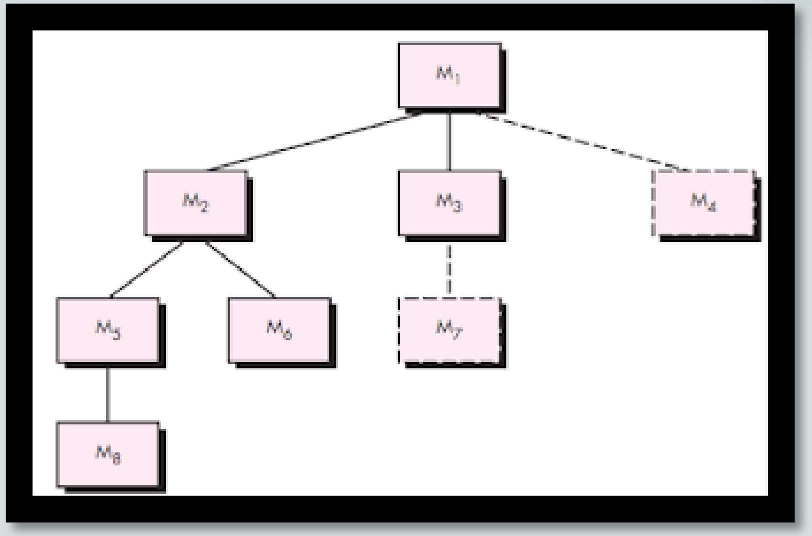
Can use **depth-first** or **breadth first** technique.

Depth first integration moves vertically to the depth, components M1, M2, M5 then M8 or M6 as demanded by M2 is added.

Breadth first integration moves horizontally through each level, after M1, components M2, M3 and M4 then M5, M6, M7 are added.

Steps :

1. Main control used as test driver and stubs are placed as subordinate modules.
2. Stubs replaced by actual modules based on depth first or breadth first integration.
3. Tests take place as each module is integrated.
4. Another stub is replaced and testing is done
5. Regression testing may be conducted.

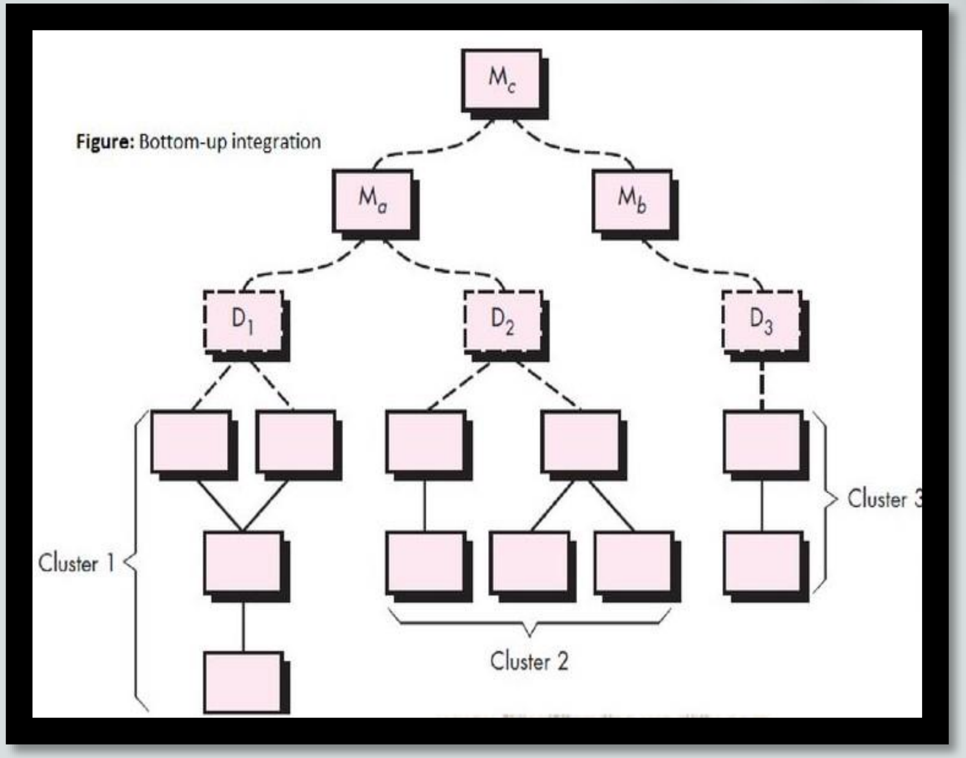


Bottom up integration

Begins at atomic modules in lower level and moves upward to main control.

Steps :

- Low level components combined to form **clusters(builds)**.
- Driver placed atop to coordinate test cases.
- Cluster is tested.
- Drivers are removed and clusters are combined.



Regression Testing

Integrating new module introduces new control logic, data flow paths and i/o functions.
This may cause problems to parts that worked correctly before.
Therefore, regression testing re-executes subset of tests that have already been done to ensure proper functioning of all components.

Different classes of test cases in Regression Test Suite :

- Test cases that checks all software functions.
- On functions that are likely to be affected by change.
- On components that have been just changed or added.

Smoke Testing

- Tests only main functionalities of a component to determine whether it works properly.
- To reveal severe errors which affect the further development of intended software.

Steps :

- Components integrated to form **builds**.
- Tests builds to uncover only show-stopper errors.
- If tests are passed, the build is integrated with other builds and are smoke tested frequently.

Benefits:

Integration risk minimized
Quality of end product improved
Error diagnosis and correction simplified
Progress is easier to assess.

Validation Testing

- Begins at the conclusion of integration testing when individual components have been tested, software completely assembled as a package, and interfacing errors have been uncovered and corrected.
- Focuses on user-visible actions and user-recognizable output from the system.

Validation-Test Criteria

Test cases that are designed to ensure

- all functional requirements and behavioral characteristics are achieved,
- content is accurate
- all performance requirements are attained

After each test case one of two possible conditions exist:

- The performance characteristic conforms to specification and is accepted.
- A deviation from the specification is uncovered and a deficiency list is created.

Configuration Review :

- An important element of validation process to ensure that all elements of the software configuration have been properly developed. It is also known as **audit.**

validation testing cont…

Acceptance Testing :

- Customer validates all requirements.
- Conducted by the end user rather than software engineers, an acceptance test can range from an informal "test drive" to a planned and systematically executed series of tests. It can span over a number of weeks or months.

Alpha and Beta Testing :

- Alpha tests are conducted at developer's site by a group of end users with the developer "looking over the shoulder" of the users by recording errors and usage problems. They are conducted in a controlled environment.
- Beta test is conducted at one or more end-user sites, developer is not present and environment is not controlled by developer. Users record problems and report to the developer who makes modifications and release software to the customer base.

System Testing

At its final stage a software is incorporated with other system elements such as hardware, people and information and is usually carried out by a team that is independent of the development team. It includes :-

Recovery Testing
Tests that forces a system to fail in a variety of ways to verify recovery is performed properly.

Security Testing
Tests whether protection mechanism built in fact protects the system from improper penetration.

Stress Testing
Tests system under conditions where it demands resources in abnormal quantity or volume.

Performance Testing
Tests run-time performance and is often coupled with stress testing.

Deployment Testing/Configuration Testing
Exercise software in environments in which it is supposed to operate and also examines installation procedures and documentation.

Black Box Testing

- Also known as Behavioural Testing, is based on testing without knowing the internal implementation/structure of the component being tested.
- Mainly focuses on functional requirements and also consider non-functional requirements to small level.
- It is done to find errors such as :
    - Incorrect/missing functions
    - Interface errors
    - Data structure / external database errors
    - Behaviour / performance errors
    - Initialization / termination errors
- Applied in Integration Testing, System Testing, Acceptance Testing

Techniques

Equivalence partitioning

Divides input values into classes of data from which test cases are derived. These *equivalence classes* contain valid and invalid partitions of the input conditions.

Boundary Value Analysis

Here test data is selected from boundaries or edges of input domain. The values at the boundary, and just above and below, or adjacent are taken.

Graph Based Testing

Cause effect graphing involves building a graph based on the causes and effect, i.e., input and output conditions and deriving test cases to cover the graph and find errors.

White Box Testing

- Other names, Glass-Box testing, Code-Based/Structural testing, Clear-Box/Transparent-Box/Open-Box Testing.
- Internal structure is known, all program based logic paths are covered.
- Test cases are derived to ensure :
    - All independent paths within the component is covered
    - True/False logic decisions are exercised
    - Loops are tested over their input domain and boundary conditions
    - Examine internal data structures for validity
- Applied in Unit Testing, Integration Testing, System Testing.

Techniques

Basis Path Testing

A logical complexity measure can be derived from these technique which acts as a basis to define independent logical paths in a module. It is done with the help of flow graph notations and then deriving paths from them and eventually test cases.

Control Structure Testing

It includes condition testing, data flow testing, loop testing

TEST STRATEGIES FOR OBJECT ORIENTED SOFTWARE

UNIT TESTING IN OO CONTEXT

- Class testing is done, units are encapsulated classes
- Classes encapsulate attributes and methods, methods are the smallest testable unit
- An operation/method can be a part of many different classes and hence cannot always be tested in isolation
- In conventional software, unit testing considers algorithmic characteristics of module and data flow among interfaces. In class testing, the operations in classes are considered and the states of classes are taken into account.

INTEGRATION TESTING IN OO CONTEXT

Two integration approaches are as follows :

Thread Based Testing

A particular input/event is taken.

Classes responding to this event are integrated into a series and tested individually.

Regression testing is applied.

Use Based Testing

Take those classes which do not collaborate much with other classes or those that use few server classes.

Lay down the classes that are dependent on the above mentioned independent classes.

This procedure is continued until the whole system is constructed.

Drivers and Stubs are used for user interface implementation and in case of collaborating classes which haven’t been implemented.

Cluster Testing is a step in integration testing where a cluster of collaborating classes are tested by specially designed test cases.

System and Validation Testing is similar in all types of software.

TEST STRATEGIES FOR WEBAPPS

Strategy for webapps can be explained using the following points :

1) *Content* model is evaluated to find errors related to syntax and semantics.
2) *Interface* model evaluated to verify all use cases are present.
3) *Design* model evaluated to uncover navigation errors.
4) *User interface* reviewed to check the standard of presentation and navigation system.
5) *Functional components* tested to verify user functional requirements are present.
6) *Navigation* throughout architecture tested.
7) *Compatibility* tested by implementing in different system configurations.
8) *Security* is tested to find vulnerabilities.
9) *Performance* is tested under various conditions and situations.
10) *Usability* is determined by end users by interacting with the webapp.

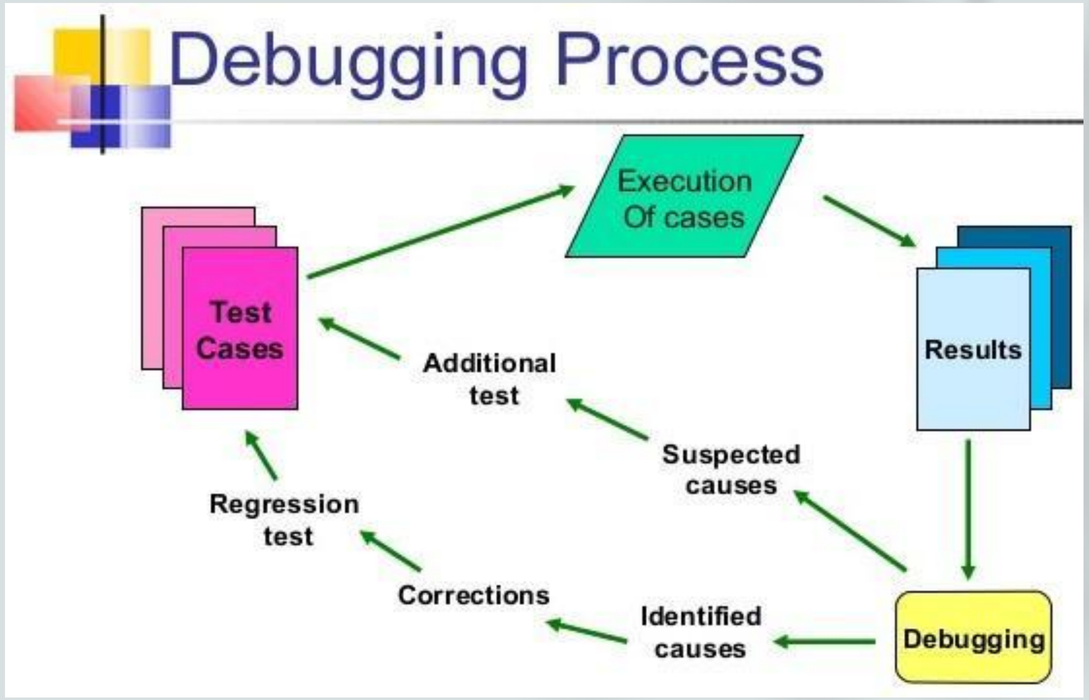
THE ART OF DEBUGGING

Debugging is the process of removal of bugs/errors in the program. One has to **identify**, **analyze** and **remove** errors once a failure occurs.

Testing is a systematic and planned activity whereas debugging is an art because many a times it cannot be done in an orderly fashion as the external view of error(referred to as *symptom*) and its internal cause location may be unrelated.

Debugging is considered difficult because :

- the symptom and cause location may be remote
- the symptom is cleared when another error is corrected
- symptoms may not be occurring regularly
- symptom may be caused by non-errors such as round-off inaccuracy
- symptom may be difficult to trace because it may be caused by some human logic error
- sometimes the problem is in timing rather than in processing
- the order of input condition which raised the symptom may be difficult to produce again to track error
- there may be causes distributed over tasks running on different processors for a single software



THE DEBUGGING PROCESS

Identify, Analyze, Remove

- Begins with execution of test case
- Result assessment is done
- ***Identify*** deviation from expected outcome which forms the symptoms
- ***Analyze*** the symptom/problem to find the cause. This may lead to :

I. cause is found

II. cause is not found, suspect a cause, design test case to validate the suspicion

- ***Remove*** the problem by correcting the cause

DEBUGGING STRATEGIES

1) BRUTE FORCE METHOD

- Most common, least efficient
- Go through memory dumps, run time traces, output statements

Try to find cause from this overload of information

2) BACKTRACKING

- Commonly used in small programs
- Begin where the symptom was uncovered

- Trace source code backward until cause is found

3) CAUSE ELIMINATION

- Uses either induction or deduction
- In both, data related to error is organized and potential causes are found
- A set of hypothesis is devised and the data are used to prove or disprove a hypothesis
- When the cause is found, the bug is isolated and corrected

AUTOMATED DEBUGGING

- Debugging tools are available for automated support in debugging
- IDEs provide ways to capture language specific errors
- Debugging compilers, dynamic debugging aids(tracers), automatic test case generators, cross reference mapping tools are available

CORRECTING ERROR

Before correcting an error

- Check whether the cause is reproduced anywhere else
- Assure it doesn’t introduce new errors
- Analyze what could have done to prevent the error so that it doesn’t occur in future