ST

Quality Assurance: Process of ensuring quality of a product or service with regard to specified requirements.

× SQA : Means of monitoring software development to ensure quality of software.

× QA : Avoiding defects prior to product development. (Assure)
× QC : Detecting defects after product is built. (Control)

× Testing : Process of discovering errors & defects in a software.
- Verification : Is the product being built right?
- Validation : Is the right product being built?

Verification : Checking / testing of items for conformance & consistency with a specification associated w/ the software.

Validation : Checking

× Testing : Primarily done upon completion of product development.

- Error : Conceptual mistake made by developers or programmers.
- Fault : Specific manifestation of an error, conforming to a logical mistake, leading to an error.
- Failure : Inability of a component / system to perform its required function within specified limits. Occurs upon execution of a fault.
- Bug : Probable faults of system.

Testing — :

× Aim to show incorrectness. Succeeds upon discovery of an error.
× Process of system evaluation by manual or automatic means to check conformance of requirements & expected & actual

results.

× Activities carried out to ensure final product meets requirements intended to be satisfied.

× Performed to detect errors, not to prove the product is error-free.

× Successful tests uncover undetected errors.

× Attributes of a good test —:
- Has high probability of discovery of an error.
- Non-redundant.
- Best of breed : Has best quality out of possible tests (best as per product requirements).
- Neither simple, nor complex.

× Involves creation of test cases to break the system.

× Principles —:
- Must be based on user requirements.
- Testing time & resources are limited.
- Impossible to test everything.
- Use effective resources to test: domain experts, users, constraints.

+ Debugging : Process of rectifying errors & bugs.
: Only performed by insiders with design knowledge.

- Test planning must be done early. (during as early as requirement phases)
- Test for invalid & unexpected conditions, as well as valid conditions to ensure conformance.
- Probability of existence of more errors in a module or group of modules is directly proportional to # errors already found.
- Testing must begin at module level (unit testing).
- Must be performed by independent party. (to avoid bias).
- Assign best personnel to the task (effective reqmts).

- Must not be planned w/ assumptions that no errors will be found.
- Software must be kept static during testing.
- Test cases & results must be properly documented.
- Provide expected results, if possible.

× Characteristics of Software Testing —:

- Testability : How easily a program may be tested. (Simple I/O).
- Operability : Better working implies more efficiently testable
- Observability : What you see is what you test.
- Controllability : Better controllability of software implies higher automation & optimization of testing.
- Decomposability : Control over scope of testing to independently test isolated components.
- Simplicity : Less functionality implies ease in testing.
- Stability : Fewer changes implies fewer disruptions to testing.
- Understandability : More information allows for smarter testing.

× Additional Principles —:
- Tests should be traceable to user requirements.
- Should begin from small (unit) to large (system, acceptance).

× Set of activities planneable in advance & conducted systematically.
× For testing, formal review of requirements is collected.
× Varying testing methods are applicable at various points of time.
× Testing conducted by developer & an independent test group.

× Test Case : Specification of input, execution conditions,

x Unit Testing : Testing indiv. programming units / entities (modules)
Modules : Smallest prog. entities.
: Essentially a set of path tests to examine various paths into the modules.
: Conducted to ensure all paths are solid & error-free & will not cause abnormal termination or other undesirable effects.

x Integration Testing : Testing multiple modules working together (post integration)
Types : Top-Down : Root Module to Sub Levels.
Adv. : High priority modules tested early to develop prototypes.
Disadv. : Errors at low module level discovered later.
Adv. : Partially working model ready to deploy.
Disadv : Limits upflow of info., limiting testing of top level.
Bottom-Up : Low Level to High Level
Adv : Through testing of modules, critical modules tested first
Disadv : Prototype not ready until multiple modules tested
Big-Bang : Testing multiple modules together in ||, all together.
Adv : Quick, low development time.
Disadv : Nothing to demonstrate until testing completes.

x Functional Testing : Tests devised to test each function separately.
: Verifies that app does what it is supposed to & does not do what it should not do.
: Works in view of requirements.
: May be difficult due to :
- Fx. within modules may consist of lower level fx, each of which must be tested first.
- Functionality may not coincide with module boundaries blurring differences b/w unit & integration testing.

- **Regression Testing** —: Process of executing a set of previously executed functional & integration tests to ensure that program changes have not degraded the system.
  - × Problems must not resurface.
  - : Problems arise when errors introduced by addition of new components affect previous tests.

- **System Testing** : Concerned w/ execution of tests to evaluate system w.r.t. user requirements.
  - : Validates system for meeting functional & non-func. reqmts
  - : Checks for unexpected interaction b/w unit & modules & evaluates system for functional reqmts.
  - : Series of tests whose primary purpose is to fully exercise the computer-based system.

- **Acceptance Testing** : Process of executing test cases agreed with the customer as being an adequate representation of user reqmts.
  - : Based on initial requirements specified by the user.

- **Recovery Testing** : Uses TC designed to examine the system in case of a disaster (disk crash, power failure, OOM, BSOD, etc)
  - : Desirable to have a system recoverable quickly w/ minimal human interaction & logs of system action prior to disaster & logs of failure.
  - : Checks how easily & completely a system may recover from a disaster.

- **Security Testing** : Testing system in order to make sure that unauth. personnel or other systems cannot gain access to information, resources or the system itself.

: Protects from improper penetration, verifies protection mechanism built into system protects from improper penetration. Tester plays role of pen tester / attacker.

Purpose : Identify & remove software flaws that may potentially lead to security violations & validating effectiveness of security measures.

× Performance Testing : Check s/w does not consume inf. resources & takes non inf time, and is free from performance bugs.
: Involves monitoring perf. levels & recording during normal, low & high stress loads. (normal = regular stress load).
: Designed to test runtime perf. of s/w within context of an integrated system.
: Occurs throughout all steps of testing process.

× Reliability Testing : Reliability: Failure-free operation of system (Probability of)
: Tests to ensure reliability.

× Robustness Testing : Robustness : Degree to which s/w can function in presence of unexpected inputs or stressful conditions.
: Tests to ensure functioning of system in robust manner.

× Stress Testing : Encompasses creation of unusual loads on system in attempt to break the system.
: Executes system in a manner where system demands resources in abnormal quantity or volume.
: Tests whole system instead of s/w alone.
: System processed to beyond operatable limits.

ST

- Load Testing : Subjecting system to a statistically representative load to determine executability
  : Varied load from min to max level. that a System can sustain w/o running out of resources or having operations suffer.

- Thread Testing : Suitable for testing real-time systems.
  : Processing threads identified & executed. Executing threads examined through simultaneous execution of tests.

- Seeding Testing : Testing via standard testing strategies using random inputs to determine overall errors.

- Defect Testing : Testing to identify defects present in system prior to delivery (presence of faults).

- Interface Testing : Discover defects in interface or medium.

- Alpha Testing : Testing by developers.
- Beta Testing : Testing by users of software, to conform to dynamic requirements of users.
- Installation Testing : To ensure that product is packaged correctly & successfully installable as per instructions in the installation document.

ST

Testing & Debugging —:

- Debugging : Aimed to find & correct errors (post testing)
- Testing : Systematic process (define TC, specify conditions, obtain results, compare/eval. against expected results).
- Debugging occurs as a consequence of successful testing (only upon discovery of error). Process for removal of errors.
- Debugging process begins with execution of TCs. (Errors discoverable upon execution of TC).
- Two-step procedure :
  - Identify cause of error : One of two outcomes
    - Cause found & corrected.
    - Cause not found.

  Suspected Cause : Reason, possible suspect, may not be discovered.
  Identified Cause : Cause discovered, & rectified.

- Testing vs. Debugging :—:

Testing	Debugging
- Process by which we find errors & bugs.	- Process by which we correct bugs found during the testing process.
- Display of errors.	- Deductive process.
- Identify failure of implemented code. (in process)	- Provide absolution to code failure (in process).
- Done by testers (possibly indep.)	- Done by developers / insiders / programmers
- No need of design knowledge.	- Requires sophisticated knowledge (insiders)
- Manual or automatic.	- Manual only. (Cannot be automated).
- Based on diff. lvls. of testing.	- Dependent on types of bugs.
- Begins after code is written.	- Begins w/ execution of TC.
- Part of SDLC (integral component).	- Not an aspect of SDLC, occurs as a consequence of testing.

× Difficulty in debugging —:
- Symptom & cause may be geographically remote: Symptom at one location, cause at another.
- Symptom may (temporarily) disappear when correction or occurrence of another error.
- Symptom may occur due to non-errors (discrepancy).
- Symptom may be due to human errors which are hard to determine.
- Input conditions may be hard to accurately reproduce.
- Symptom may be due to causes distributed across tasks running over multiple processors (different).
- Symptom may be a result of timing problems, rather than processing problems.
- Symptom may be intermittent.

× Debugging Techniques —:

× Brute-Force Debugging:
- De-facto, common, least-efficient method, used when other techniques fail.
- Performed by addition of print statements in different parts of code so as to expect to find cause based on output results.
- Program loaded w/ print statements to display intermediate values w/ hope that some printed values will help to identify statements w/ error.

× Backtracking Debugging:
- Involves backtracking incorrect results through logic of program.
- Source code traced backwards until error is discovered.
- Unfit when # of source lines (LOC) to be traced increases & # potential bkwd. paths become unmanageable.

× Debugging by Induction:
- Process that locates data, organizes it & devises a hypothesis. Hypothesis must then be proved.

× Debugging by Deduction:
- Determine possible causes, & use data to eliminate them.
- Refine remaining cause into a hypothesis & prove the same.

× Cause-Elimination Debugging :
- List of causes which may have contributed to the error symptom is prepared. Tests then carried out to eliminate each cause.

× Debugging by Testing :
- Involves use of two kinds of test cases :
  - Cases which expose a previously undetected error.
  - Cases which provide useful info. in debugging to locate an error.

× Debugging by Program Slicing :
- Overall search space first divided into program slices, so that search is confined to program slice only.
- Similar to backtracking.

Test Cases —:

× Document containing set of test data, preconditions, expected results & postconditions developed for a particular test scenario in order to verify compliance against a specific requirement.
× Set of actions executed to verify a particular feature / functionality of a software application. Low-level actions.
× Required to check if particular app or s/w is working or not.
× Defines how to test a system, s/w or app.
× Set of actions performed on a system to determine if it satisfies s/w requirements & functions correctly.
× Advantages of effective test cases:
- Guaranteed good test coverage.
- Reduced maintainance & s/w support cost.
- Improved quality of s/w & user experience (UX).
- More reliability of produced products
- More satisfied customers owing to high quality.
- Increased profits owing to satisfied customers.

× Characteristics of effective test cases :
- Simple & clear.
- Traceable : aids in discovery of error;
- Minimum description :
- Zero Assumptions :
- Unique :
- Concise :
- Transparent :
- Understandable :
- Easy to implement :
- Able to handle varying input :
- Meets requirements of software :
- Returns repetitive results :

× Parameters of test cases :
- Module name :
- Test case ID :
- Test scenario : scenario for execution of the TC.
- TC description :
- Steps of the TC :
- Pre-requirements :
- Post-requirements :
- Test data :
- Expected results :
- Actual results :
- Status :
- Comments :

Black-Box v/s White-Box Testing —:

Black-Box Testing	White-Box Testing
Testing approach used to test s/w w/o knowledge of internal structure of programs / apps.	Testing approach where internal structure, program or code is known to the tester.
Test engineers perform this testing (developers may too).	Developers only may perform this testing.
It is known what the s/w is supposed to do, but no knowledge of how is available or relevant.	Apart from knowledge of what the s/w is supposed to do, knowledge of how is also a must to be known.
Functionality of application is verified based on the app. requirement (SRS) specification.	Source code is examined & logic of the program is tested for correctness.
To perform this testing, no need of an understanding of the used programming languages	To perform this testing, knowledge & understanding of used programming language is required.
No need of understanding of the internal code design.	Understanding of the internal design of the code is necessary.
Alternatively known as Data-driven testing, Box testing, & Data & Functional testing.	Alternatively known as Structural testing, Clear-Box testing Code-Based testing & Glass-Box testing.
Can be applied virtually to every level of s/w testing. (Unit, Integration, System, Acceptance)	Can be applied mainly at units testing, but at times at system & integration levels as well.
Test design techniques : - Decision-table testing - Equivalence partitioning	Test design techniques : - Control-Flow testing - Branch testing & Path testing

- Boundary value Analysis - Cause-Effect Graph Technique	- Statement Coverage - Decision Coverage
× Deals with higher levels of testing (System, Acceptance)	× Best suited for lower levels of testing (Unit, Integration)
× Implementation knowledge not required	× Complete understanding of implementation is required.
× Programming Language knowledge is not required.	× Programming Language knowledge is required
× Difficult to automate.	× Easier to automate.
× Eg : β Testing	× Eg : α Testing.
× Main objective : check what functionality of system is undertested	× Main objective : check quality of program / code.
× Can be performed by end-users, developers & testers.	× Usually done by testers & developers.
× Can start after preparation of requirement specification document.	× Can start only after preparation of detail design document.
× Less exhaustive, time-consuming	× Exhaustive & time-consuming
× Does not require code access	× Access of code is required.
× Not suitable / best for algorithm testing	× Best for algorithm testing. (Logic knowledge required)
× Efficient for large code segments	× Allows removal of extra LOC which may bring in hidden defects.
× Low-skilled testers may perform this testing.	× Expert testers w/ vast experience & programming knowledge required.
× Functionality of s/w can be tested w/o going deep into the code.	× Tester detects all logical & design errors into the code.

ST

Black-Box Testing —:

- Specification / Behavioral / Data-Driven / Function (Focus on action) / Input-Output Testing.
- Attempts to uncover: incorrect functions, data structure errors, missing functions, performance errors, initialization & termination errors, external DB access errors.
- Unaware about internal functionality, so testing possible by any independent tester.
- Advantages: Fast (focus on I/O), easy, cost-efficient, unbiased (test cases not as per code), no requirement of programming knowledge at tester end, testing in perspective of user, test cases designable as soon as requirements established.
- Disadvantages: Difficulty pinpointing cause of error, difficult to design test cases.
- Techniques

× Equivalence-Class Partitioning:

- Divides input into classes of data, where equivalence classes are classes where each member represent a valid or invalid value.
- Test cases designed for both valid & invalid values, where result of valid test case expected to be correct.
- Test cases derived for classes of data obtained from division of program input.
- Eg: Input Range: 0-100.

  Valid Range = 1TC, (Eg = 25-50) Invalid Range = 2TC (min range & max range) (Eg = 0-24 & 51-100)

  : Specific Value: Valid = 1TC, Invalid = Other values (1TC)

  : Set of Values: Valid = 1TC (member of set)
  Invalid = 1TC (member of complement)

  : Boolean Value: Valid = 1TC (valid input), Invalid = 1TC (complement)

ST

White-Box Testing — :

- Test cases selected on basis of code instead of specifications.
- Types :

× Basis Path Testing — :
- Allows design & definition of a basis set of execution paths.
- Examines each path of the code by executing it at least once.

× Structural Testing —:
- Examines source code, and analyzes what is present in it.
- Often dynamic, implying code is executed during testing
- Subtypes :
  + Statement Coverage Testing : Ensures every stmt (conditionals) is executed at least once.
  + Branch Coverage Testing : Testing of all branches at least once. Requires sufficient TCs for each program decision, to ensure all possible outcomes are generated at least once.
  + Conditional Coverage Testing : Testing of all conditions in program to generate all possible outcomes at least once.
  + Loop Coverage Testing : All loops to be tested for one, two, zero & multiple iterations.
  + Path Coverage Testing : All paths are tested at least once. Capable of uncovering more faults than branch testing. Requires sufficient TC to test every path from start to end to be tested at least once. Due to large # of paths, # of paths tested depend on risk involved (all paths cannot be tested).

(1) Suitable testing technique dependent on use case or scenario : user / time / resource considerations.
  + Domain & Boundary Testing : Path domains are subtypes of program input for execution of unique paths. Input data can be derived from CFG. Test inputs are chosen to

- exercise each path as well as boundary conditions / cases for the domain.

+ Data Flow Testing : Focuses on points variables receive values and the points at which the values are used. Studies sequences & actions of variables at program paths.

× Logic-Based Testing — :

- Used when input domain & resulting processing are amenable to a decision table.
- Steps : List all actions executed in the algorithm
  : List all conditionals evaluated during algorithm execution
  : Associate specific conditions w/ specific actions.
  : Define rules to determine which action corresponds to which condition, so as to discover gaps & inconsistencies easily.

× Fault-Based Testing — :

- Shows absence of certain classes of faults in code.
- Subtype : Mutation Testing : Introduce mutants (variations) in code to discover unexplored errors in code. Useful for large industry scenarios.

Cyclomatic Complexity : # of paths
: # of regions +1 (outer region)
: e - v + 2
: p (predicate nodes) +1