SOFTWARE TESTING

GENERAL DEFINITIONS

- **Software Testing**
  Any activity aimed at evaluating an attribute or capacity of a program or system and determining that it meets its required results.

- **Software Quality** is the degree of conformance to explicit or implicit requirements and expectations.
    - Explicit: clearly defined and documented
    - Implicit: not clearly defined and documented but indirectly suggested
    - Requirements: business/product/software requirements
    - Expectations: mainly end-user expectations
  - ***Software Quality Dimensions***
    - Accessibility
    - Compatibility (different Operating Systems, Browsers, etc.)
    - Concurrency (multiple requests to the same resources at the same time.)
    - Efficiency (achieve a result without wasted energy, resources, effort, time)
    - Functionality (to carry out the functions as specified or desired)
    - Installability (to be installed in a specified environment)
    - Localizability (to be used in different languages, time zones etc.)
    - Maintainability (the ease with which software can be modified)
    - Performance (the speed at which software performs under a particular load)
    - Portability (to be transferred easily from one location to another)
    - Reliability (to perform a required function under stated conditions and time without any errors)
    - Scalability (measure in performance in response to changes in software's processing demands)
    - Security (protection of software)
    - Testability (to be easily tested)
    - Usability (the degree of software's ease of use)

  *** *If smo says "This software is of a very high quality.", you might ask "In which dimension of quality?"*

- **Quality Assurance (QA)**; is a set of activities for ensuring quality in software engineering processes (that ultimately result in the quality of software products)
  Activities: Process definition and implementation, Auditing, Training
  Processes include: Software Development Methodology, Project Management, Configuration Management, Requirements Development/Management, Estimation, Software Design, Testing, etc.

- **Verification vs Validation**

Criteria	Verification	Validation
Definition	The process of evaluating work-products (not the actual final product) of a development phase to determine whether they meet the specified requirements for that phase.	The process of evaluating software during or at the end of the development process to determine whether it satisfies specified business requirements.
Objective	To ensure that the product is being built according to the requirements and design specifications.	To ensure that the product actually meets the user's needs and the specifications were correct.
Question	Are we building the product right?	Are we building the right product?
Evaluation Items	Plans, Requirement Specs, Design Specs, Code, Test Cases	The actual product/software.
Activities	• Reviews • Walkthroughs • Inspections	• Testing

SOFTWARE TESTING

- **Software Development Models**
    - Waterfall Model
    - Spiral Model
    - Iterative and incremental development
    - Agile development

- **SDLC (Software Development Life Cycle)**
    1. Requirement gathering and analysis
    2. Design
    3. Coding
    4. Testing
    5. Deployment
    6. Maintenance

- **STLC (Software Testing Life Cycle)**
    1. Requirement / Design Analysis
    2. Test Planning
        - Test Plan
        - Test Estimation
        - Test Schedule
    3. Test Case Development (Designing)
        - Test Cases / Test Scripts / Test Data
        - Requirements Traceability Matrix
    4. Test Environment Setup
    5. Test Execution
        - Test Results (Incremental)
        - Defect Reports
    6. Test Closure Activity (Reporting)
        - Test Results (Final)
        - Test Metrics
        - Test Closure Report

SOFTWARE TESTING

SOFTWARE TESTING LEVELS

1. **Unit Testing**
   Tests units of code using automation to ensure that each unit works individually; its integration into the system as a whole is tested later.
   - First level of software testing.
   - *WHEN PERFORMED:* **Unit T**--> Integration T--> System T--> Acceptance T
   - *METHOD*: It is performed by using White Box Testing method.
   - *WHO PERFORMED*: It is normally performed by software DEVELOPERS (rarely by independent software testers)

2. **Integration Testing**
   Determines if multiple components work together properly within the system.
   - *WHEN PERFORMED:* Unit T--> **Integration T**--> System T--> Acceptance T
   - *METHOD:* Any of Black Box Testing, White Box Testing and Grey Box Testing methods can be used. Normally, the method depends on your definition of 'unit'.
   - *WHO PERFORMED*: Developers themselves or independent testers

3. **System Testing**
   Analyzes the behavior of the whole system and whether or not it fits defined requirements.
   - *WHEN PERFORMED:* Unit T--> Integration T--> **System T**--> Acceptance T
   - *METHOD:* Usually, Black Box Testing
   - *WHO PERFORMED*: Normally, independent testers

4. **User Acceptance Testing**
   Verified that it can handle real-world scenarios and is often a contractual requirement for acceptance of the software; performed by actual end users of the system.
   - *WHEN PERFORMED:* Unit T--> Integration T--> System T--> **Acceptance T**
   - *METHOD:* Usually, Black Box Testing. Testing does not normally follow a strict procedure and is not scripted but is rather ***ad-hoc.***
     - **Ad-hoc Testing (Random Testing, Monkey Testing)**:
       - Conducted informally without any planning and documentation
       - The tester improvises the steps and arbitrarily executes them (like a monkey typing while dancing)
   - *WHO PERFORMED*:
     - Internal Acceptance Testing (Alpha Testing) is performed by members of the organization who are not directly involved in the project (Development or Testing).
     - External Acceptance Testing: is performed by people who are not employees.
       - Customer Acceptance Testing
       - User Acceptance Testing (Beta Testing)

SOFTWARE TESTING

SOFTWARE TESTING METHODS

1. **Black Box Testing (Behavioral Testing)**
   Examines the functionality of an application *without knowledge* (so named black box) of internal structures.
   - Can be usually FUNCTIONAL or NON-Functional
   - Generally, Software Tester is responsible
   - Basis for test cases is Requirement Specifications
   - Mainly applicable to higher levels of testing (System T and Acceptance T)
   - ***TECHNIQUES***;
     - ***Equivalence CLASS Partitioning:*** It is a software test design technique that involves dividing input values into valid and invalid partitions and selecting representative values from each partition as test data.
       You can not test everything. You can not test whole grape basket before buy.
       Class the thing like; Rich people, men, age…
     - ***Boundary VALUE Analysis:*** It is a software test design technique that involves the determination of boundaries for input values and selecting values that are at the boundaries and just inside/ outside of the boundaries as test data.
       Your credit card spending limit 5000 and you should at least spend 50 to avoid monthly fee.
       50 is **Lower Boundary**. 5000 is **Higher Boundary** or higher limit
       - Proactive Approach (We are) - quality assurance
       - Active Approach
     - ***Cause-Effect Graphing:*** It is a software test design technique that involves identifying the cases (input conditions) and effects (output conditions), producing a Cause-Effect Graph, and generating test cases accordingly.

2. **White Box Testing**
   Tests internal processes of an application, as opposed to its functionality.
   - Generally, Software Developer is responsible
   - Mainly applicable to lower levels of testing (Unit T and Integration T)
   - Basis for test cases is Detail Design
3. **Gray Box Testing**
   - is a combination of Black Box Testing method and White Box Testing method.
   - the internal structure is partially known.
   - It is primarily used in Integration Testing

4. **Agile Testing**
   - Welcome changes

5. **Ad-Hoc Testing (Random, Monkey)**
   - Is a method of software testing without any planning and documentation.
   - This method is normally used during Acceptance Testing.
   - Conducted informally and randomly without any formal expected results.
   - The success of ad hoc testing depends on the creativity and tenacity of the tester (and, of course, luck)

SOFTWARE TESTING

SOFTWARE TESTING TYPES

1. **Smoke Testing (Build Verification Testing)**
   A quick and simple test to ensure that the major functions of a software work as intended, originally considered successful when new hardware didn't catch fire.
   - Name comes from the device passed the test if it did not catch fire (or smoked) the first time it was turned on.
   - If the smoke test passes, go ahead with further testing.
   - Smoke testing is normally used in Integration T, System T and Acceptance T.
   - Do NOT consider smoke testing to be a substitute of functional/regression testing.

2. **Functional Testing**
   Assesses functional components of the system and ensures that it performs the basic functions that is tested against the functional requirements/specifications.
   - Ensures that the requirements are properly satisfied by the application.
   - Black Box Testing technique is used in which the internal logic of the system being tested is not known to the tester.
   - Is normally performed during the levels of System T and Acceptance T.
   - Functional testing is more effective when the test conditions are created directly from user/business requirements.

3. **Usability Testing**
   Technique which evaluates an app on how easy it is for use it.
   - is normally performed during System T and Acceptance T levels.
   - is NOT to be confused with User Acceptance T or User Interface / Look and Feel T.

4. **Security Testing**
   Determines the safety of private data without compromising a system's functionality.
   - Network Security
   - System Software Security
   - Client-side Application Security
   - Server-side Application Security
   - is not the only (or the best) measure of how secure an application is. But, it is highly recommended that security testing is included as part of the standard software development process.

5. **Performance Testing**
   An umbrella term that includes all processes, which assess the user's experience with the system. It intends to determine how a system performs in terms of responsiveness and stability under a certain load.
   - **Load T**; conducted to evaluate the behavior of a system at increasing workload.
   - **Stress T;** conducted to evaluate the behavior of a system at or beyond the limits of its anticipated workload.
   - **Endurance T;** conducted to evaluate the behavior of a system when a significant workload is given continuously.
   - **Spike T;** conducted to evaluate the behavior of a system when the load is suddenly and substantially increased.

6. **Regression Testing**
   Ensures that changes to the code haven't broken any previously-functional parts of the system.
   - Regression: the act of going back to a previous place or state; return or reversion.
   - can be performed during any level of testing (Unit, Integration, System, or Acceptance) but it is mostly relevant during System Testing.
7. **Compliance Testing (Conformance, Regulation, Standards Testing)**
   - to determine the compliance of a system with internal or external standards.

SOFTWARE TESTING

1. **Compatibility Testing**
   Ensures functionality of the system across a variety of browsers and operating systems.

2. **Mobile Testing**
   Ensures that the application functions across a variety of mobile devices.

3. **Accessibility Testing**
   Verifies that your product is accessible to customers who have disabilities.

4. **Localization Testing**
   Ensures that software performs as required in different countries, and that translations are correct.

5. **Live Testing**
   Tests functions and features that can only be accurately tested on the live system instead of a test environment.

6. **Static Testing**
   Examination of the system's code and documentation, without running it.

TESTING DOCUMENTATION (ARTIFACTS)

1. **Test Plan**

   Describes the scope, approach, resources, schedule, test items, features to be tested and not tested, tasks, and contingencies for the entire testing process. Types;

   - Master Test Plan
   - Testing Level Specific Test Plans (Unit TP, Integration TP, System TP, Acceptance TP)
   - Testing Type Specific Test Plans; like Performance TP and Security TP

   TEMPLATE

   a. **Test Plan Identifier**
   b. **Introduction**
   c. **References**
   d. **Test Items**
   e. **Features to be Tested**
   f. **Features Not to be Tested**
   g. **Approach**
   h. **Item Pass/Fail Criteria**
   i. **Suspension Criteria and Resumption Requirements**
   j. **Test Deliverables**
   k. **Test Environment**
   l. **Estimate**
   m. **Schedule**
   n. **Staffing and Training Needs**
   o. **Responsibilities**
   p. **Risks**
   q. **Assumptions and dependencies**
   r. **Approvals**

2. **Test Case**

   A set of inputs, preconditions, predicted results and execution conditions for a particular objective or test condition.

Test Suite ID	The ID of the test suite to which this test case belongs.	TS001
Test Case ID	The ID of the test case.	TC001
Test Case Summary	The summary / objective of the test case.	To verify that clicking the Generate Coin button generates coins.
Related Requirement	The ID of the requirement this test case relates/traces to.	RS001
Prerequisites	Any prerequisites or preconditions that must be fulfilled prior to executing the test.	1. User is authorized. 2. Coin balance is available.
Test Procedure	Step-by-step procedure to execute the test.	1. Select the coin denomination in the Denomination field. 2. Enter the number of coins in the Quantity field. 3. Click Generate Coin.
Test Data	The test data, or links to the test data, that are to be used while conducting the test.	1. Denominations: 0.05, 0.10, 0.25, 0.50, 1, 2, 5 2. Quantities: 0, 1, 5, 10, 20
Expected Result	The expected result of the test.	1. Coin of the specified denomination should be produced if the specified Quantity is valid (1, 5) 2. A message 'Please enter a valid quantity between 1 and 10' should be displayed if the specified quantity is invalid.
Actual Result	The actual result of the test; to be filled after executing the test.	1. If the specified quantity is valid, the result is as expected. 2. If the specified quantity is invalid, nothing happens; the expected message is not displayed
Status	Pass or Fail. Other statuses can be 'Not Executed' if testing is not performed and 'Blocked' if testing is blocked.	Fail
Remarks	Any comments on the test case or test execution.	This is a sample test case.

Created By	The name of the author of the test case.	John Doe
Date of Creation	The date of creation of the test case.	01/14/2020
Executed By	The name of the person who executed the test.	Jane Roe
Date of Execution	The date of execution of the test.	02/16/2020
Test Environment	The environment (Hardware/Software/Network) in which the test was executed.	OS: Windows Y Browser: Chrome N

Characteristics of a good test case:

- Accurate: Exacts the purpose.
- Economical: No unnecessary steps or words.
- Traceable: Capable of being traced to requirements.
- Repeatable: Can be used to perform the test over and over.
- Reusable: Can be reused if necessary.

3. **Test Script**
   Specifies test procedures, typically for automated testing.
   A test script can be as simple as the one below:

```
def sample_test_script (self):
type ("TextA")
click (ImageButtonA)
assertExist (ImageResultA)
```

TESTING TOOLS

- **Bug Tracker**
  Keeps track of software bugs, usually made up of an automatic flow of bugs, a set of customizable reports, and different system roles and permissions.
- **Test Management System**
  Stores testing info from the test planning stage, through execution of cases, up to the reporting stage.
- **Application Lifecycle Management System**
  Helps a company manage its software development life cycle in a dedicated, centralized place.
- **Automation Tools**
  Control the execution of tests and the comparison of actual outcomes with predicted outcomes.
- **Performance Testing Tools**
  Measure an app's response while simulating the traffic of tens or even hundreds of thousands of users performing a wide range of tasks on the app.

TESTING STRATEGIES

- **Risk Based Testing**
  Prioritizes test processes based on the risk their flaws pose to an app, and the effect those flaws could have on the company or end-user.
- **Exploratory Testing**
  Test design and test execution at the same time a sophisticated, thoughtful approach to ad hoc testing.
- **Automated Testing**
  Uses coded scripts to control the execution of tests without manual intervention.
- **Continuous Integration**
  Members of a team integrate their work frequently, usually each person integrates at least daily-leading to multiple integrations per day.
- **Requirement Traceability Matrix (RTM)**
  Maps test cases to requirements to ensure testing coverage of requirements.

SOFTWARE TESTING

DEFECT

Defect Management Process

1. **Discovery** ->
2. **Categorization** ->
   a. *Severity / Impact*
      - *Describes the **seriousness** of defect*
      - *Can be defined as the degree of impact a defect has on the development or operation of a component application being tested*
      - Critical (shut-down of the process)
      - High / Major (collapse the system, however, certain parts remain functional)
      - Medium / Minor (some undesirable behavior)
      - Low / Trivial

   b. *Priority / Urgency*
      - *Describes the **importance** of defect*
      - *States the order in which a defect should be fixed.*
      - P1 (High)
      - P2 (Medium)
      - P3 (Low)

		SEVERITY	
		Critical	**Non-Critical**
PRIORITY	**Urgent**	Key Feature does not work	Company Logo is the wrong color
	Low	Feature that is rarely used does not worked	The caption on an image is written in the wrong font

   c. *Probability / Visibility*
      - *High (encountered by all/almost all the users of the feature) / Medium / Low*

   d. *Related Dimension of Quality*

   e. *Related Module / Component*
   This provides information on which module / component is buggy or risky.

   f. *Phase Detected*
   Indicates the phase in the software development lifecycle where the defect was identified. (Unit T, Integration T, System T, Acceptance T)

   g. *Phase Injected*
   Phase Injected is always earlier in the SDLC than the Phase Detected. Phase Injected can be known only after a proper root-cause analysis of the bug.
      - Requirements Development
      - High Level Design
      - Detailed Design
      - Coding
      - Build/Deployment

3. **Resolution** ->
   After receiving the defect from the testing team, DEV team conduct a review meeting to fix them. Then they send a resolution type to the testing team for further communication. Resolution types:
   - Accept
   - Reject
   - Duplicate
   - Enhancement
   - Need more info
   - Net reproducible

- Fixed
- As designed

4. **Verification** ->
5. **Closure** ->
6. **Reporting**

   After uncovering a defect (bug), testers generate a formal defect report.

Defect Life Cycle (Happy Path)

STATUS	
NEW	When a defect has been logged
OPEN / ASSIGNED	When the DEV accepts the defect and starts working on it
RESOLVED / COMPLETED	DEV has fixed the issue
RETEST	QA will make sure if the defect is actually fixed
CLOSED / VERIFIED	QA successfully retested the defect
RE-OPEN / REASSIGNED	When Retest has failed
REJECTED / DROPPED	When QA & DEV are in agreement
CANCELLED	When QA agrees that defect was wrongly logged