Frequently Used Concepts in Software Testing

ver 0.5

A

- **Acceptance Testing:** Testing that checks whether the system meets the business requirements and is acceptable to end users.
  (Typically performed just before releasing the software.)
- **Accessibility Testing:** Testing to ensure the software is usable by all users, including people with disabilities (such as blindness, low vision, deafness, or mobility impairments).
  (For example, verifying compliance with accessibility standards like WCAG.)
- **Ad Hoc Testing:** Software testing done without any pre-planned scenario or documentation. It's completely informal and improvised, manipulating and exploring the system to uncover unexpected bugs.
- **Alpha Testing:** Testing performed internally by the development teams before the official release.
- **API Testing:** Directly testing an API (either automated or manual) to verify that correct and incorrect inputs are handled properly and that the expected outputs are returned.
- **Assertion:** The expected result or behavior that the code should satisfy or not. (For instance: Assert.equals(4, calculator.add(2, 2)))
- **Automated Testing:** Using scripts or software tools to run tests automatically, making them faster, more accurate, and free from human error.

📌 Example: In an online store, instead of manually logging in and making a purchase each time, use Selenium or Playwright to write an automated test that fills out the login form, selects products, completes payment, and checks the outcome.

B

- **Baseline:** A standard or initial reference point against which later tests are compared. This helps determine if a change has led to improvements or not.
- **Beta Testing:** Testing the software by actual (selected or volunteer) users in real-world conditions, but before the official release.
- **Black Box Testing:** Testing performed without knowledge of the internal structure, code, or other additional information—based only on inputs and outputs. In this approach, the tester behaves like a regular user, checking whether the system responds as expected.
  (Often used in security testing.)
- **Bottom-Up Testing:** An approach where testing begins with the lower-level modules (e.g., standalone functions or classes) and gradually moves up to higher-level modules (more complex modules and the entire system). When a module isn't ready yet, a driver is typically used to fill its place.

📌Example: In a banking app, test the loan interest calculation module first. Once it's verified, test the loan registration module in the database, and finally test the entire loan request process end to end. 🔄✅

- **Boundary Value Analysis:** Testing the software with values at the edges (boundaries) of input domains.

📌Example: If a field must accept values from 1 to 100, test with 0, 1, 99, 100, and 101.

- **Build Verification Test (BVT):** A set of quick, critical tests of the system's main capabilities. If these tests fail, there's no need to spend time on deeper tests because the build is fundamentally flawed.

C

- **Code Coverage:** A metric that shows what percentage of the code is covered by tests. A higher percentage means more parts of the code are executed during testing. However, a high percentage alone doesn't guarantee good tests—test quality also matters.

📌Example: 80% code coverage means 80% of the total code has been executed in tests. Methods or conditionals not covered reduce this percentage.

C

- **Compatibility Testing:** A crucial non-functional test checking whether the software works correctly across various platforms, browsers, devices, and configurations. It ensures that users, no matter their system, get a consistent, problem-free experience.
- **Concurrency Testing:** A type of performance testing that checks whether the system functions correctly under the simultaneous use of multiple users or processes.
- **Configuration Testing:** A type of non-functional testing that examines how the software runs under different system settings and configurations. It ensures the application is compatible with various hardware combinations, operating systems, drivers, and server configurations.
- **Continuous Integration (CI):** Code is continuously merged into the repository, and every change is automatically tested and verified. With CI, each new code addition triggers a build and automated tests. If any problem arises, it's detected quickly, making bug fixes faster.

📌 Example: Using Jenkins, GitLab Runner, or GitHub Actions to run automated tests after every push to the repository.

- **Cross-Browser Testing:** Testing web applications in different browsers to ensure the layout and behavior remain consistent across all supported browsers.

D

- **Data-Driven Testing:** Tests where inputs come from data sources (e.g., files, databases).

📌 Example: Running a login test with hundreds of username-password combinations taken from a CSV file.

- **Decision Table Testing:** First, enumerate all possible combinations of inputs and conditions, then verify the system's correct behavior under each combination. This method is especially useful when there are multiple conditions and business rules.
- **Dependency Testing:** A key part of integration testing that checks whether interdependent modules in a software system function together properly without errors.
- **Destructive Testing:** Testing the software by intentionally subjecting it to abnormal, extreme, or destructive conditions to see how resilient it is and how it reacts. This type of testing is crucial for evaluating the system's stability and security.
- **Dynamic Testing:** One of the two main categories of software testing (along with static testing). It examines code while it's running to see if the actual output matches the expected outcome. This type of testing typically happens after coding is done and can be manual or automated.

E

- **End-to-End Testing (E2E Testing):** Testing that simulates real scenarios from start to finish, ensuring the entire system (and all of its components) works correctly as a whole. (Unlike unit testing, which tests only a small part of the system.)

📌 Example: Testing the full purchase flow in an online store, from searching for a product to completing the order.

- **Equivalence Partitioning:** Dividing all possible input values into several groups (partitions) so you don't have to test every single value; you just test representative samples from each group. This makes tests fewer but more strategically chosen.
- **Exploratory Testing:** Testing the system by exploring and manipulating it without strictly predefined scenarios, aiming to discover hidden issues or bugs. It relies on the tester's experience and curiosity rather than strictly scripted procedures.

I

- **Impact Analysis:** Evaluating which parts of the software might be affected or broken by a given change or update. This helps determine which areas need re-testing whenever a modification is introduced.
- **Integration Testing:** Testing that ensures the various parts of the software work correctly once they're integrated. It checks interactions between modules, services, or components to verify that data and processes flow properly.
- **Interface Testing:** Checking how two parts of the software (e.g., two services, an API and frontend, or a database module and backend) communicate with each other. This ensures the input and output between these components are correctly transferred.

K

- **Keyword-Driven Testing:** Tests that run based on keywords and conventions rather than direct coding. This allows testers to build and run complex automated tests without extensive programming skills.

📌 Example: In a testing tool, instead of writing `click("LoginButton")`, you might write "Click Login" and the tool automatically knows to click the login button. Common in setups like Selenium + Robot Framework. 🤖✅

L

- **Load Testing:** Evaluating how a system behaves under normal or slightly heavier-than-usual traffic. It shows whether a server, database, or application continues to function under high usage or breaks down.

📌 Example: A ticket-selling site where 10,000 people try to buy tickets in one hour. Load Testing simulates these conditions to see at what point the site slows or crashes. 🚦🔥

- **Localization Testing:** Checking whether the software is correctly localized for a specific region or language, including translations, date formats, currency, and layout direction.

📌 Example: When localizing an app for Iran, ensure it uses the Persian calendar correctly, that pages are right-to-left, that prices are shown in rials, and that the translations feel natural.

M

- **Maintainability Testing:** Assessing how easy it is to modify, update, and debug the software. This test focuses on code structure, documentation, and architecture to see how straightforward future maintenance and development will be.

📌 Example: An app with clean, modular, and well-commented code gets a good maintainability rating; if everything is tangled and undocumented, even a small change can be a big headache. 🛠️🔍

- **Manual Testing:** Testing the software without using automated tools: the tester checks different parts of the system manually to see if there are any bugs. This approach doesn't scale or repeat as easily as automated tests.
- **Mocking:** Creating a "fake" version of a code dependency that just pretends to act like the real thing without actually doing the real work. This is common in unit testing so that tests don't rely on external services like databases or APIs. However, poorly done mocking can mask real-world errors.

📌 Example: If there's a `SendEmail()` method, in tests you create a Mock that only checks "Was the email call triggered?" without actually sending an email.

- **Model-Based Testing (MBT):** Designing and running tests based on a mathematical model or diagram describing system behavior. That model could be a state machine, flowchart, or data model that automatically generates test scenarios.

N

- **Negative Testing:** Deliberately feeding invalid, unexpected, or strange inputs to the system to see how it reacts. The goal is to ensure it doesn't crash and returns the correct errors in abnormal conditions.

📌 Example: In a sign-up form, instead of a normal name, try inputs like numbers, `!@#$%^&*`, or very long text. The system should produce the correct error message rather than crashing. 💥🔍

N

- **Non-Functional Testing:** Testing aspects of the software related not to "what the system does" but "how it does it"—such as performance, security, user experience, and stability. It checks whether the system just "works" or if it "works well."

📌 Example: For an online shopping site, functional testing checks if an order can be placed, while non-functional testing looks at how fast the order is processed, how it handles thousands of concurrent users, and how secure it is. 🔐✅

P

- **Pair Testing:** When two people (usually a tester and a developer) work together on testing to quickly discover bugs and fix them on the spot. This approach can make testing more effective and deepen understanding of the system.

📌 Example: A QA engineer and a developer sit together, run manual or automated tests, and if a bug is found, the developer immediately checks the code and fixes the issue. 👨‍💻🤝👩‍💻

- **Penetration Testing ⌴ PenTest:** Security testing that simulates a real attack on the system to find vulnerabilities before malicious hackers can exploit them. Typically carried out by ethical hackers or security experts to see whether sensitive data can be accessed.
- **Performance Testing:** Assessing software in terms of speed, stability, and scalability under load or stress. It measures response time, resource usage, and system behavior under heavy loads.
- **Production Testing:** Testing the software directly in the real (production) environment without negatively impacting actual users. Often used to verify performance, stability, and potential issues post-release.

📌 Example: Enabling a new feature (with a feature flag) for only 5% of users, observing whether it works properly, then gradually rolling it out to more people.

Q

- **Quality Assurance (QA):** A set of processes and standards that help produce software with minimal bugs, high quality, and alignment with user needs. QA isn't just about testing—it includes planning, preventing defects, improving the development process, and ensuring code quality.

📌 Example: In a software team, a QA Engineer doesn't just run tests but also reviews code, defines testing standards, refines CI/CD processes, and steers the team toward a high-quality product from the outset.

- **Quality Control (QC):** Checking and testing the software after development to ensure it meets requirements and quality standards. Unlike QA, which focuses on processes, QC focuses on detecting issues or defects in the final product.

📌 Example: After building a new app, the QC team tests all features, checks the user interface, and verifies outputs to ensure they match what's expected.

R

- **Regression Testing:** After every change or update, retesting to verify that old (already working) features are still functioning correctly. It ensures that a fix or new feature doesn't break something else.

📌 Example: If you fix a bug in the payment system, run regression tests to ensure it hasn't caused discounts or coupons to stop working! 🔄✅

- **Reliability Testing:** Testing how stable the software is over time or under specific conditions. It reveals how long the system can run without crashes, errors, or degraded performance.
- **Requirements Testing:** Checking whether the software has been implemented according to specified requirements. It ensures all expected features are present and functioning correctly.
- **Risk-Based Testing:** Prioritizing testing efforts on the parts of the software that carry the highest risk of failure or user impact. Instead of testing everything at the same level, critical areas receive the most focus.

📌 Example: In a banking application, a bug in the profile settings page isn't as catastrophic as a bug in online payments, so risk-based testing emphasizes payment and security features more. ⚠🔍

S

- **Sanity Testing:** A brief, surface-level test performed after a small change or bug fix to confirm that the issue is resolved, without running deeper or broader tests. It's like a quick check before regression testing.
- **Smoke Testing:** A quick, initial test to verify if a new build is stable enough for more extensive testing. Usually focused on essential, critical features. If smoke tests fail, the build is deemed too broken to warrant further testing.
- **Static Testing:** One of the two main categories of software testing (along with dynamic testing) performed without executing the code, focusing on reviewing documents, code, and software structure beforehand. This helps catch issues early in the development cycle.
- **Stress Testing:** A performance test that checks how the system behaves under extreme, unusual, or overload conditions (e.g., high traffic, resource constraints, or service interruptions). It determines the software's resilience.
- **Stub:** In integration and unit testing, a simplified module or method that replaces a real dependency in a test. A stub usually returns fixed data without complex logic.

Difference between Stub and Mock:
- Stub only returns test data.
- Mock simulates the behavior and interactions of a class.

- **System Testing:** Testing the entire software system as one integrated piece to verify that all parts work correctly together. It happens after integration testing and before acceptance testing.

T

- **Test Case:** A detailed, predefined scenario designed to validate a specific feature. Each test case outlines what to test, which inputs to use, and what the expected output is.
- **Test-Driven Development (TDD):** Writing tests before writing the actual code, leading to cleaner, more testable code with fewer unexpected bugs.

TDD cycle:
1️⃣ Write a test that initially fails ❌ (because we have no implementation yet)
2️⃣ Write the minimal code needed to make the test pass ✅
3️⃣ Refactor and improve the code 🔄
4️⃣ Run the tests again to confirm correctness ✅

- **Test Environment:** A setup similar to the production environment but dedicated to software testing. It usually includes a test database, test server, test APIs, and monitoring tools so tests can run without affecting real users.

Example of a banking app's test environment:
✅ A separate server hosting the trial version of the app 🖥️
✅ A test database filled with fake data 🗄️
✅ A test payment API that doesn't withdraw real money! 💰🚫

- **Test Harness:** In automated testing, a collection of tools, scripts, and frameworks used to run tests, check outputs, and report results. They let you automate complex tests and simulate conditions like varied inputs, stress tests, or interactions between modules.
- **Test Plan:** A document outlining the strategy, objectives, resources, schedule, and scenarios for testing. It specifies what to test, how to test it, who will run the tests, and which tools will be used.
- **Test Script:** A coded or scripted procedure to automate a particular test. Typically written with tools like Selenium, JUnit, xUnit, or Cypress, replacing manual steps with automated ones.
- **Test Suite:** A collection of test cases or test scripts that collectively verify a specific area of software or the entire system. They can be manual or automated and often run together to perform comprehensive testing on a feature.

T

- **Top-Down Testing:** In integration testing, you start by testing the high-level modules first, then move to the lower-level ones. If some lower-level modules aren't ready yet, you replace them with stubs (temporary code that simulates their behavior).

U

- **Unit Testing:** One of the most fundamental types of software testing, where each small portion of the code (usually a single function or method) is tested in isolation.
- **Usability Testing:** A non-functional test that examines how easy, understandable, and user-friendly a piece of software or a website is. Usually done with real users.
- **User Acceptance Testing (UAT):** One of the final stages of software testing to check whether the product truly meets the business needs and end-user expectations. It's commonly performed by actual users or business representatives, who confirm if the software is ready to be released.

V

- **Volume Testing:** Testing how the system behaves with a large volume of data. It helps determine if the software continues to function correctly or becomes slow and unstable when the database or server is filled with huge amounts of data.

W

White Box Testing: A testing approach that examines the internal structure, logic, and data flows in the code. Unlike Black Box testing (where only outputs are observed), White Box requires insight into the code's internals and designs tests based on its logic.

https://www.linkedin.com/in/aminmesbahi/
https://x.com/aminmsbh
https://mesbahi.net