**Git**

**Purpose:**

* Tracks changes in code
* Allows for simultaneous collaboration without affecting each other's code
* Enables changes without damaging the master code
* Facilitates goal setting and project management with special tools
* Identifies files by content, not by name

**Unit Test:**

**Purpose:**

* Tests a specific function in a class
* Other tests are a series of actions
* Unit test should be a class

**Characteristics:**

* Written by the developer themselves

**Mocking:**

* Simulates the behavior of a specific object
* Example: Testing a function that promotes a user from a regular user to an admin. Instead of performing the entire registration process, prepare a set of users in advance and use them in the function.
* Prepare the data before the action within the function

**Assertion:**

* Helps understand the expected result and whether the result failed or not, then check where the failure occurred to fix it.

**Test Driven Development:**

* Know in advance the functions to be written, expect the result since the tests are known, and then write the function

**Python Testing Libraries:**

**unittest:** Most popular in Python, built-in

**pytest:** Needs to be installed, more advanced

**Structure:**

* **Module:** A Python file that contains what should be tested
* **Unit test:** A class named Test\_Module\_Name
* Methods inside the class are named test\_<name\_of\_function\_from\_module>

**Process:**

1. Prepare the environment
2. Perform the action
3. Make an assertion
4. Cover all possibilities
5. Ensure clean and readable code and maintain it

**Note:** No input in tests as it may stop the automatic process.

**CICD:**

* Process in the software world after the testing phase. It takes the entire application and tests it in stages.

**Automatic Test Structure:**

* Do not write tests that check multiple things; each test should check one thing
* **Autonomous Test:** Independent and focused. Example: The test itself selects a product from the site and adds it to the cart, manages its information without needing another test to intervene.
* **Atomic Test:** Checks specific things defined within it.

**Complexity:**

* Complexity of a test, many steps complicate and require more maintenance

**Execution Time:**

* Shorten times as much as possible

**Fragility:**

* The test may break along the way; prepare things in advance

**Issue Identification:**

* If the test is short, it's easy to identify where the problem is. Long tests with 50 steps, for example, are hard to debug, so split them.

**Limited Isolation:**

* In small tests, it's very easy to isolate the problem because it checks one thing.

**Feedback Delay:**

* It's easy to identify problems when there are small tests, so feedback will be easier.

**Isolated Tests:**

**Advantages:**

* Short run time
* Easy to maintain
* Easy to find bugs
* Early problem identification

**Disadvantages:**

* Hard to understand how much coverage we have
* Many tests
* Doesn't cover many cases
* Requires time and resources

**Main Point:** Small and independent tests

**AAA: Arrange, Act, Assert**

**Process:**

1. **Arrange:** Prepare the data, for example, select a product
2. **Act:** Perform the action, for example, add the product to the cart
3. **Assert:** Comparison action, check the result

**Act:** Includes the action that affects the assert

**Setup & Teardown:**

**Setup:**

* Prepare data once, and then the tests use this data
* **Before all:** Define data before all tests
* **Before each:** Define data before each test

**Teardown:**

* Clean the data added, return a clean environment
* **After each:** Clean after each action
* **After all:** Clean after all actions, return a clean environment

**Autonomous and Atomic Tests in QA Automation:**

**Autonomous Tests:**

* Independent of one another, crucial for:
  + Parallel Execution: Can be executed in parallel, reducing total run time
  + Reliability: A test failure doesn’t impact other tests, making issue identification easier
  + Maintenance: Easier to maintain since changes in one test don’t affect others
  + Scalability: Test suite can grow without interference among tests

**Atomic Tests:**

* Verify a single functionality or behavior in isolation:
  + Simplicity: Each test has a clear, simple purpose
  + Isolation: Runs independently without complex setup/teardown
  + Clarity: Easy to understand and maintain
  + Precision: Failures provide precise information about application issues

**Differences and Overlaps:**

* Scope: Autonomous tests focus on independence, atomic tests on granularity
* Interdependency: An atomic test can be autonomous, but not all autonomous tests are atomic
* Design Philosophy: Autonomous tests avoid dependencies, atomic tests isolate functionality

**Best Practices:**

* Combine both principles
* Modular setup and teardown
* Clear purpose for each test

**Manual vs. Automated Tests:**

**Manual Tests:**

* More flexible
* Often have a human element

**Automated Tests:**

* Performed by computers, follow specific scripts
* Each step is considered a test
* Ensure specific coverage scenarios

**Test Failures:**

1. **Test Failure**
2. **Code Failure**
3. **Definition Failure:** The test script itself
4. **Verification Failure:** Final result doesn’t match expectations

**False Flag:**

* Incorrect error message; verify expected messages
* Prevent this by being specific in requirements
* Indicates a test passed incorrectly

**False Positive:** Test passed, but the received message didn’t match expectations

**False Negative:** Test failed due to an issue in the test script, not the code

**Note:** False positive is worse than false negative

**API vs. UI:**

**API Testing:**

* Backend or server-side components
* Focuses on functionality, reliability, performance, and security
* Faster, more stable, better coverage, early issue detection, easier CI/CD integration

**UI Testing:**

* Frontend or client-side components
* Ensures UI elements function correctly
* Tests user experience, visual verification, interactivity
* Slower, more prone to breakage, requires frequent updates, resource-intensive

**When to Use:**

* **API Testing:** Core logic, business rules, workflows, performance, and data integrity
* **UI Testing:** User workflows, visual correctness, end-to-end testing, user actions

**Sleep & Retries:**

**Sleep:** Time between iterations