Fundamentals Review

- **Quality Assurance (QA):** A systematic process to ensure that the quality of the software meets the required standards and specifications.
- **Quality Control (QC):** A subset of QA that focuses on identifying defects in the actual products produced.
- **Testing:** The process of executing a program or application to identify any gaps, errors, or missing requirements in contrast to the actual requirements.

- **SDLC & STLC full lifecycle steps**

SDLC Phase	Description	STLC Phase	Description
Requirement Analysis	Gathering and analyzing requirements from stakeholders.	Requirement Analysis	Understanding the testing requirements based on specifications.
Planning	Defining the scope, resources, and timeline for the project.	Test Planning	Defining the scope, approach, resources, and schedule for testing activities.
Design	Creating architecture and design specifications for the software.	Test Case Design	Creating detailed test cases and test scripts based on requirements.
Implementation	Writing the actual code based on design specifications.	Test Environment Setup	Preparing the environment where testing will be conducted.
Testing	Verifying that the software meets the requirements and is free of defects.	Test Execution	Running the test cases and documenting the results.
Deployment	Releasing the software to the production environment.	Defect Reporting	Logging any defects found during testing.
Maintenance	Ongoing support and updates to the software after deployment.	Test Closure	Evaluating cycle completion criteria and finalizing testing documentation.

- Software testing principles (e.g., testing types, test levels)

    - **Testing Shows Presence of Defects:** Testing can show that defects are present, but cannot prove that there are no defects.
    - **Exhaustive Testing is Impossible:** It is impractical to test all possible inputs and scenarios.
    - **Early Testing:** Testing should start as early as possible in the SDLC.
    - **Defect Clustering:** A small number of modules usually contain most of the defects.
    - **Pesticide Paradox:** Running the same set of tests will not find new defects; tests must be regularly reviewed and revised.
    - **Testing is Context Dependent:** The testing approach should be tailored to the specific context of the project.
- Agile vs Waterfall model – key points

Aspect	Agile Model	Waterfall Model
Approach	Iterative and incremental development.	Linear and sequential development.
Flexibility	Highly flexible; changes can be made at any stage.	Less flexible; changes are difficult to implement once a phase is completed.
Phases	Phases overlap; development and testing occur simultaneously.	Distinct phases; each phase must be completed before the next begins.
Customer Involvement	Continuous customer collaboration and feedback throughout the project.	Limited customer involvement after the requirements phase.
Documentation	Minimal documentation; focuses on working software.	Extensive documentation required at each stage.
Testing	Testing is integrated throughout the development process.	Testing occurs only after the development phase is complete.
Delivery	Frequent delivery of small, functional increments.	Delivery of the complete product at the end of the project.
Best Suited For	Projects with evolving requirements and a need for rapid delivery.	Projects with well-defined requirements and low likelihood of changes.
Team Structure	Cross-functional teams with collaborative roles.	Defined roles with a clear hierarchy.
Risk Management	Continuous risk assessment and adaptation.	Risk is assessed primarily at the beginning of the project.

- Bug Life Cycle

  The Bug Life Cycle describes the various stages a bug goes through from identification to resolution. Here are the typical stages:

  1. **New**: The bug is identified and logged.
  2. **Assigned**: The bug is assigned to a developer for fixing.
  3. **Open**: The developer starts working on the bug.
  4. **Fixed**: The developer has fixed the bug and it is ready for testing.
  5. **Retest**: The QA team retests the bug to verify the fix.
  6. **Verified**: The bug is confirmed as fixed.
  7. **Closed**: The bug is closed if no further issues are found.
  8. **Reopened**: If the bug still exists after retesting, it is reopened for further investigation.

Real-Time Examples:-

- Bug fixed after developer cleared cloud cache: A UI issue was resolved after the developer cleared the cloud cache, as stale data was affecting the latest changes.
- Bug due to code not committed: A feature appeared broken during testing because the developer had forgotten to commit the updated code.
- Bug due to error in code logic: A functionality bug was traced back to incorrect logic implemented in the code, which was fixed after debugging.
- Bug due to wrong API usage: The feature failed due to the use of an incorrect API endpoint, which was corrected by pointing to the right one

Sample Bug
Title: Login Successful Without Email Verification. [P1]

Summary: [Engg] Unverified users were able to log in successfully and access the dashboard, bypassing the mandatory email verification step.

Steps to Reproduce:

1. Register a new user with a valid email and password.
2. Do not verify the email (skip the verification link).
3. Go back to the login page.
4. Enter the same credentials and click "Login".

Expected Result: User should be blocked from logging in and shown a message like: “Please verify your email before logging in.”

Actual Result: User was logged in successfully and redirected to the dashboard, gaining full access.

PFA logs and screenshots.