1. What is Software? Types of Software with Example

Software is more than just a computer program.
It includes:

- **Programs** (the code that runs),
- **Procedures** (how to use it),
- **Documentation and Data** (manuals, help files, settings, etc.)

It helps the computer system to work and perform tasks.

Types of Software

There are **two major types** of software:

1. Generic Software (Buy)

- This type is **already developed** and sold to many people.
- It is called **Commercial Off-The-Shelf software (COTS)**.
- Same software is used by thousands of users.
- It may not work exactly how you want it to.
- **Example:** Microsoft Word, Adobe Photoshop

2. Customized Software (Build)

- This is made **specifically for one user or organization**.
- It meets special needs of that customer.
- It is also called **Bespoke software**.
- **Example:** A bank may get special software built just for its internal operations.

Buy & Build Together

- Most of the time, personal users use **COTS software**.
- But companies may **buy and then customize** it for their needs.

2. What is Software Quality?

- According to standards:
  - **ISO/IEC 9126:** Software quality means how well a product meets user needs.
  - **IEEE Std 610:** It's about meeting both requirements and user expectations.

1. Functionality

Does the software do what it's supposed to do?

- Measures **how well** the software performs its required tasks.
- Includes things like correctness, security, and interoperability.

- **Example:** A banking app should correctly transfer money between accounts.

2. Reliability

Can the software be trusted to work under different conditions?

- Tells us how **dependable** the system is.
- How often does it fail? Can it recover from errors?
- **Example:** Autopilot software must not crash during flight, even if there's a minor fault.

3. Usability

Is the software easy to use and understand?

- Focuses on **user experience** – ease of learning, using, and navigating the system.
- **Example:** A mobile app with a clean interface and intuitive icons is more usable.

4. Efficiency

Does the software use system resources wisely?

- Looks at **performance** – like response time, memory usage, etc.
- **Example:** A photo editing app that runs smoothly without slowing down your device is efficient.

5. Maintainability

Can the software be easily fixed, updated, or improved?

- Concerns how **easy it is to modify** the system after it's deployed.
- **Example:** If you can easily fix a bug or add a feature without breaking other parts, that's maintainability.

6. Portability

Can the software work on different devices or environments?

- Measures how easily it can be transferred from one hardware or software environment to another.
- **Example:** A website that works on Chrome, Firefox, mobile, and desktop has high portability.

Q2. What are the challenges in software projects? Explain.

Answer:

In software projects, the main goal is to deliver the product **on time**, **within budget**, and with the **expected quality**. But achieving this is not easy. There are several challenges that make software development difficult.
The main challenges are:

1. Time

- Projects are often **delayed** and not completed on time.
- Managing schedules and meeting deadlines is a big issue.

2. Cost

- Many software projects become **too expensive**.
- They go **over budget** due to poor planning or unexpected problems.

3. Scope (Quality)

- Sometimes, the **final product does not meet quality expectations**.
- It may have **many bugs or missing features**.
- This happens due to unclear requirements or lack of proper testing.

Summary
These three major issues—**time**, **cost**, and **quality**—are known as the **Project**

Management Triangle.
Balancing all three is the biggest challenge in software developme

Give examples of software defects.

Answer:

A software defect is a problem or error in the software that causes it to behave unexpectedly or fail. Defects can be **very dangerous and costly**, especially in critical systems.

Here are two real-life examples of software defects:

1. Therac-25 Radiation Machine (1986)

- It was a **radiation therapy machine** used to treat cancer.
- Due to a **software defect**, the machine gave patients **massive overdoses of radiation**.
- This caused **serious injuries and deaths**.
- The bug was due to poor testing and no proper safety checks in the software.

2. Ariane 5 Rocket Failure (1996)

- The **Ariane 5 rocket** exploded just **40 seconds after launch**.

- Cause: A **software bug** in the navigation system.
- The system tried to convert a large number into a small memory space, which caused a crash.
- The failure cost **over $370 million**.

Conclusion:

These examples show that **software defects can lead to loss of money, property, and even human lives**. So, quality assurance and testing are very important in software development.

Q4. What is Software Testing? What are its goals and levels?

Answer:

What is Software Testing?

Software testing is the process of **executing a program** to find **errors or bugs**.
It checks if the software works correctly and meets user requirements.

Goals of Software Testing:

1. **To check if the software meets its requirements.**
   – Make sure the software does what it's supposed to do.
2. **To find errors and bugs.**
   – Testing helps locate problems so they can be fixed early.
3. **To test performance in real situations.**
   – For example, testing how an autopilot system works under stress or emergency.

Levels of Software Testing:

There are **3 main levels** of testing:

1. Unit Testing

- Tests **individual parts** or modules of the software.
- Example: Testing a single function like “login”.

2. Integration Testing

- Checks if **different modules work together** properly.
- Example: Does the login screen connect correctly to the user database?

3. System Testing

- Tests the **complete software as a whole**.
- It checks the full system behavior and performance.

Conclusion:

Software testing is a key part of software quality. It helps deliver **reliable and correct software** to users by finding and fixing problems early.

Q5. What is the role of testing in software development?

Answer:
Testing plays a very important role in software development. It helps ensure that the software is **correct, reliable, and ready for use**.

There are **two main types of testing activities** based on how the testing is done:

1. Static Analysis (Without Running the Program)

- In this method, we **check the code manually** or using tools without running it.
- This includes:
  - **Code reviews**
  - **Walkthroughs**
  - **Formal inspections**
- It is **manual and time-consuming**, but good for finding logical or design errors early.

2. Dynamic Analysis (Running the Program)

- In this method, the program is **executed with input values** to check its behavior.
- It helps find **runtime errors** like crashes, wrong output, etc.
- It is usually **automated and faster**, but may not catch all errors.

Best Practice: Use Both

- To get the best results, **both static and dynamic analysis** should be used.
- This way, we can **catch more defects** and improve software quality.

Conclusion:

Testing helps to find bugs early, reduce risks, and improve the **overall quality of the software**. It is a **key activity** in every phase of software development.

Q6. What is Software Quality Assurance (SQA)?

Answer:
Software Quality Assurance (SQA) is a set of **planned and systematic activities** that make sure the software **meets quality standards and works properly**.

It is not just testing — SQA covers the **entire software development process** to make sure the product is being built correctly at every stage.

Main Points about SQA:

1. **Planned & Structured**
    - SQA is done using a proper plan and follows rules, methods, and standards.
2. **Monitoring the Process**
    - It checks whether the team is following the correct steps while building the software.
3. **Like an Umbrella**
    - SQA covers **all phases**: requirement, design, coding, testing, and release.
    - It includes everything from **process checks to reviews and audits**.
4. **Goal**
    - To make sure that the final software product is **high-quality**, **meets user needs**, and is **free of defects**.

Conclusion:

SQA helps in building **reliable and quality software** by focusing on **process improvement** and **early detection of issues** during development.

Q7. What is Software Quality Control (SQC)?

Answer:
Software Quality Control (SQC) is the process of **checking whether the software product follows the quality plan** made during Software Quality Assurance (SQA).

It focuses on **reviewing the product** at different stages to find and fix problems before release.

Main Activities in SQC:

1. **Review of Requirements**
    - Checking if user needs are clear and complete.
2. **Design Reviews**
    - Making sure the software is designed properly and can be built correctly.
3. **Code Reviews**
    - Reading the source code to find bugs or bad practices.
4. **Testing the Product**
    - Running the software to catch any defects or failures.
5. **Deployment Review**
    - Ensuring the final version is ready and stable for use.

Goal of SQC:

- To **verify that the product is correct** and matches the original plans and user requirements.

Conclusion:

While **SQA focuses on improving the process**, **SQC focuses on checking the actual product**. Both are important for building high-quality software

Q8. Differentiate between Software Quality Assurance (SQA) and Software Quality Control (SQC).

Answer:

SQA	SQC
Software Quality Assurance	**Software Quality Control**
Focuses on the **process** used to create the software.	Focuses on the **product** that is being developed.
It is **preventive** — aims to **avoid defects** by improving the process.	It is **detective** — aims to **find defects** in the product.
Done **throughout the software development life cycle**.	Done **after or during product development** (e.g., during testing).
Activities include **planning, process monitoring, audits, reviews**.	Activities include **reviews, testing, inspections**.
Example: Ensuring developers follow coding standards.	Example: Testing the final software for bugs.

Conclusion:

- **SQA** ensures **quality is built into the process**.
- **SQC** ensures **quality is present in the final product**.
  Both are necessary for delivering high-quality software.

Validation vs. Verification — What's the Difference?

Term	Meaning	Focuses On	Keyword
Validation	Are we building the **right** product?	Meeting **user needs**	**Requirements**
Verification	Are we building the product **right**?	Meeting **specifications**	**Correctness**

🔎 **Validation**

- Checks **if the software meets customer's expectations**
- Focuses on **what** the software is supposed to do
- Done **after development** or **at the end of each phase**

🧠 **Think:**
👉 Are we solving the user's problem?
👉 Does this feature really help the user?

📌 **Example:**
A university result system should display **GPA with subject-wise marks**.
If it only shows GPA, **it fails validation** — the user needs more.

🧪 **Verification**

- Checks if the product was **built correctly**
- Focuses on **how** the system is made (code, logic, specs)
- Done **during development**

🧠 **Think:**
👉 Did we implement the function properly?
👉 Are the outputs matching the expected ones?

📌 **Example:**
In a billing system, if total = price × quantity,
but the code does **price + quantity**,
then it **fails verification** — the logic is incorrect.

🔁 **In Short:**

Validation	**Verification**
Are we building the **right** product?	Are we building the product **right**?
Focus on **user needs**	Focus on **technical specs**
Ensures **usefulness**	Ensures **correctness**
Late-stage process	**Early-stage process**

💡 **Mnemonic to Remember:**

"Validation = Vision" (user's vision/needs)
"Verification = Function" (checking functions/logic)

❗ Difficulties in Achieving Good Quality

Achieving good software quality isn't always easy. Several factors can make it harder for developers to maintain quality throughout the development cycle. Let's look at each of them:

1️⃣ Size (Lines of Code)

- **What it means:**
  As software grows larger, it becomes harder to test. Specifically, the more lines of code (LOC) you have, the more complex the system gets.
- **Why it's difficult:**
  With **manual testing**, it's hard to spot logical errors in huge codebases. Imagine you have thousands of lines of code, and it's almost impossible to manually go through every line to check for mistakes.
  Example: If a website has hundreds of pages, testing for broken links manually is very time-consuming.

2️⃣ Invariant Complexity in the Product and Project

- **What it means:**
  Some software systems are very complex because they involve a lot of logic that has to work correctly. Even a small mistake in the logic can cause **big problems**.
- **Why it's difficult:**
  Software like autopilot systems or medical devices must follow strict logic. If the logic is wrong, it can result in dangerous outcomes. **Example:** If the autopilot system incorrectly sets the plane's landing mode, it could lead to a crash.

3️⃣ Environmental Stress/Constraints

- **What it means:**
  Software must often work under **high stress** or heavy load, such as handling lots of users or data at once.
- **Why it's difficult:**
  Testing for maximum load or usage is difficult. Software might work fine with a few

users, but what if thousands or millions try to access it at once? **Example:** A **result publication website** might crash during peak times when many students check their results at the same time.

4️⃣ Flexibility/Adaptability Expected

- **What it means:**
  Software needs to be flexible and able to adapt to new requirements or changes over time.
- **Why it's difficult:**
  When the software changes, **regression testing** must be done to make sure old features still work. But frequent changes can lead to additional testing, and each change adds more complexity. **Example:** After adding a new feature, like a "Dark Mode" on an app, you need to test whether all existing features still work as expected.

5️⃣ Cost and Market Conditions

- **What it means:**
  The cost of testing, especially with **automated tools**, can be very high. Additionally, market conditions might push for faster releases, which could reduce the time allocated for proper testing.
- **Why it's difficult:**
  Automated testing tools often require **licenses** and setup costs. There's also pressure to release the product quickly, leading to a rushed testing process. **Example:** To test a new mobile app, you might need specialized tools, but if the budget is tight, manual testing might be the only option.

📝 Summary of Challenges:

Difficulty	What It Means	Example
Size (LOC)	Large codebase is hard to test fully	Testing a website with thousands of pages
Logic Complexity	Complex logic with big consequences	Autopilot software failing to land safely
Environmental Stress	Testing under high load or stress	A website crashing during result release
Flexibility	Frequent changes need retesting	Adding "Dark Mode" and retesting old features

Cost & Market Pressure	High cost of automated testing	Limited budget for testing and tools

Software Quality Engineering (SQE)

Software Quality Engineering (SQE) is a process that ensures software quality through various activities related to **validation** and **verification**. These activities help in confirming that the software meets the required quality standards and functions as intended.

Key Activities in SQE:

1. **Planning**:
    - **Quality Planning** is about setting up strategies and methods to achieve quality goals during the software development and testing process. It defines how quality will be maintained throughout the project.
2. **Execution**:
    - This refers to **executing the selected QA (Quality Assurance) plan** and conducting **software validation and verification** activities. These activities include testing the software, reviewing designs, and verifying that the software works as intended.
3. **Decision**:
    - After executing tests, you need to **decide the pass/fail criteria** of each test. For example, determining if a software feature works as expected or if there are bugs to fix.
    - **Measurement and Analysis**: Data is collected to demonstrate the software's quality to all parties involved, providing evidence that the software meets the expectations of the customers.

Why It's Important:

- Customers and users expect software to meet their quality requirements. SQE ensures that these expectations are fulfilled by validating and verifying the software at various stages.

SQE Activities

1. **Testing**:

- The most common SQE activity. Testing is performed to **remove defects** and **ensure quality** by identifying and fixing bugs or errors in the software.
- **Example:** Testing a new app feature to check if it functions as expected before releasing it to users.

2. **Other QA Alternatives to Testing**:
    - Not all quality assurance is about testing. There are other methods to improve software quality:
    - **Defect Prevention**:
        - This activity involves planning to avoid defects before they occur. This could involve setting up processes to catch errors early in development.
        - **Example**: Writing code with checks for common errors or having peer reviews to ensure high-quality coding standards.
    - **Formal Verification**:
        - This involves activities like **Inspection**, **Review**, and **Walkthroughs**, which are formal ways of checking software for defects without running the program. These methods involve examining the software's design, code, or documentation.
        - **Example**: A team of developers reviewing a new feature in the code to ensure that it follows the required standards before implementation.
    - **Fault Tolerance**:
        - This involves designing the system to handle failures gracefully. **Fault tolerance** ensures that the system remains operational even when unexpected problems occur.
        - **Example**: Adding features like **data backup**, **autosave**, or making sure the system can recover from failures (e.g., a document autosaving every few minutes).

Summary of SQE:

SQE Activity	Description	Example
Planning	Setting strategies to achieve quality in the project.	Planning testing schedules, risk management.
Execution	Carrying out the QA plan and software validation activities.	Running tests, verifying software functions.

Decision	Deciding pass/fail criteria and analyzing measurement data.	Determining whether a feature meets requirements.
Testing	Identifying defects through testing and fixing them.	Manual or automated tests to catch bugs.
Defect Prevention	Planning to avoid defects from the start.	Coding standards, code reviews, best practices.
Formal Verification	Conducting inspections, reviews, and walkthroughs.	Reviewing code or designs without executing the software.
Fault Tolerance	Designing systems to withstand failures and recover.	Adding features like autosave or backup

Error, Fault, Failure, and Defect

These terms are often used interchangeably in software development but have distinct meanings. Let's break them down:

Error:

- **Definition**: A mistake made by a human (developer or user) that leads to incorrect behavior or results in the software.
- **Example**: A programmer incorrectly writes a function that doesn't handle certain edge cases. This is an error in the code caused by a human.

Fault:

- **Definition**: A problem introduced into the software due to an error. A fault can be a result of an incorrect step, process, or data definition in the code.
- **Example**: If a programmer writes a condition incorrectly (e.g., using the wrong comparison operator in a condition), this creates a fault in the system that can lead to failures later.

Failure:

- **Definition**: A failure happens when the software doesn't work as expected or doesn't meet the required specifications. It is the observable consequence of a fault in the software.

- **Example**: A software program crashes when a user tries to submit a form because a fault in the code causes an error when validating the form data. The program's inability to complete the task (form submission) is a failure.

Defect:

- **Definition**: A general term that refers to any error, fault, or failure in the software. Defects are often the root cause of problems in the system.
- **Example**: A bug in the code that causes the software to produce incorrect results (e.g., an arithmetic calculation error) is a defect.

Summary:

Term	Definition	Example
Error	A human mistake leading to incorrect results.	A developer misplaces a = instead of == in a condition.
Fault	An incorrect step or problem in the program caused by an error.	A fault in the logic that prevents correct form validation.
Failure	When the software doesn't meet its requirements or specifications.	A program crashing when a user tries to submit a form.
Defect	A general term for errors, faults, or failures.	A bug in the software that causes it to behave incorrectly.

Note:

- **Bugs**: In common software development jargon, defects, faults, errors, or failures are all referred to as **bugs**.

Complete Testing

Testing is a crucial part of software development, and its primary goal is to ensure that software works as expected and is free from defects (bugs). Here's a breakdown of **complete testing**:

Objectives of Testing:

1. **Detect Bugs**: Identify and fix errors or defects in the software.

2. **Reduce the Risk of Defects**: Minimize the chances that defects will be overlooked and make it into the final product.
3. **Reduce the Cost of Testing**: By detecting bugs early and efficiently, we can save costs compared to finding bugs late in the development process.

Complete/Exhaustive Testing:

- **Definition**: Complete testing means testing every possible scenario in the software to ensure no undisclosed faults (bugs) remain at the end of the testing phase.
- **Ideal Goal**: The idea is that after complete testing, there should be no defects left, and all potential issues should be caught.

Challenges of Complete Testing:

Complete testing sounds perfect, but it’s nearly impossible to achieve for most software systems due to several factors:

1. **Large Domain of Possible Inputs**:
    - **Why It's Hard**: A software system typically accepts many different inputs, both valid and invalid. The number of potential combinations of inputs can be so large that it's practically impossible to test every single one.
    - **Example**: If a program accepts user inputs like text, numbers, and special characters, trying every possible combination would require an enormous amount of testing effort.
2. **Complex Design Issues**:
    - **Why It's Hard**: If the software's design is too complex, it may have too many different paths and interactions to test. This makes it impractical to check every combination of events, decisions, or functions.
    - **Example**: In a complex e-commerce platform with thousands of products, users, and transactions, testing all possible workflows and interactions might take years.
3. **Execution Environments**:
    - **Why It's Hard**: Sometimes, creating all the possible execution environments (different hardware, operating systems, networks, etc.) in which the software could run is simply not feasible.
    - **Example**: Testing an autopilot system in every possible environment, weather condition, and location (simulated or real) is nearly impossible.

Conclusion:

While complete testing is the ideal, it’s often impractical for complex systems due to the sheer number of potential inputs, the complexity of the design, and the many different environments in which the system may operate. Instead, testing is usually done in a targeted way, focusing on the most critical features and the most likely failure points to balance risk, cost, and effectiveness.

Testing Activities

Testing is a systematic process, and to ensure its effectiveness, it is divided into clear activities. Here's a breakdown of the **steps involved in testing**:

Steps in the Testing Process:

1. **Identify the Objective to be Tested (Prioritized)**:
    - Before starting testing, you need to identify **what** you are testing and **why**. This helps prioritize the most important aspects of the software that need to be verified.
    - **Example**: If you're testing an e-commerce website, the priority could be testing the checkout process to ensure that payments are processed correctly.
2. **Select Inputs from the Input Domain**:
    - The input domain refers to all possible inputs the program can accept. Since testing all possible inputs might be impossible, you need to carefully select which inputs to use for testing.
    - **Example**: If you're testing a calculator app, you would select inputs like positive numbers, negative numbers, zero, and edge cases like large numbers.
3. **Compute the Expected Outcome**:
    - For each test case, you need to **define** what the expected result should be based on the input provided. This allows you to compare the actual outcome with the expected one.
    - **Example**: For the input "2 + 2" in a calculator, the expected outcome is "4."
4. **Set Up the Execution Environment**:
    - The program needs to run in a specific environment (e.g., operating system, version, hardware). Setting up the environment ensures that the test is executed in the same conditions the final software will run under.

- **Example**: If you're testing an app, ensure it’s running on the intended devices (e.g., Android, iOS) and the correct software versions are used.

5. **Execute the Program**:
    - This is where you **run** the software with the selected inputs. During execution, you monitor how the program behaves and observe any failures or issues.
    - **Example**: Running the calculator app with various operations like addition, subtraction, etc.
6. **Analyze the Test Results with Expected Outcome**:
    - After the program executes, compare the **actual** result with the expected result. If there’s a mismatch, it indicates a defect or failure.
    - **Example**: If "2 + 2" outputs "5" instead of "4," you have found a defect.

Subset of Input Domain and Program Behavior:

Testing doesn’t always involve covering every possible scenario but focusing on key subsets of the input domain.

1. **Divide the Input Domain (D)**:
    - The input domain DDD consists of all possible inputs the program can handle. To make testing manageable, you divide DDD into smaller, meaningful subsets.
    - **Example**: In a registration form, inputs like name, email, and password can be divided into subsets like valid inputs, invalid inputs, boundary conditions, etc.
2. **Select a Subset (D1) to Test Program (P)**:
    - Instead of testing every possible input, you select a **subset (D1)** of the input domain to focus on. This subset is often chosen based on the likelihood of the inputs causing errors or their importance in the system.
    - **Example**: Instead of testing every possible username format, you may select a few valid and invalid email formats.
3. **Subset D1 May Only Exercise Part of the Program (P1)**:
    - Testing a subset of inputs may only exercise part of the program. This means that testing one set of inputs may not reveal issues in other parts of the program.
    - **Example**: Testing the login functionality with correct and incorrect credentials may not test other features like password reset or user registration.

Summary:

Testing involves several critical steps, such as identifying objectives, selecting relevant inputs, setting up the execution environment, running the software, and comparing results. To make testing manageable and efficient, we often focus on a subset of the input domain to test the most critical or likely-to-fail parts of the system.