**1. Introduction to Software Testing**

**Explanation:**  
Software Testing is the process of evaluating and verifying that a software application does what it is supposed to do. It ensures the software is free of bugs and meets user requirements.

**Real-Time Use Case:**  
When releasing a mobile banking app, testing is performed to ensure that users can log in, view their balance, and transfer money without errors. If the login or transaction fails, it can lead to customer dissatisfaction or financial loss.

**2. Software Testing - Definitions**

**Explanation:**

* **Verification:** Are we building the product right?
* **Validation:** Are we building the right product?
* **Bug/Defect:** A flaw in the software that causes incorrect results.
* **Test Case:** A set of actions executed to verify a particular feature or functionality.

**Real-Time Use Case:**  
In an e-commerce site, if a user adds a product to the cart but it doesn’t reflect in the checkout page, this defect is reported using a test case like "Add product to cart".

**3. Need of Software Testing**

**Explanation:**

* To deliver quality products
* To prevent defects
* To gain customer trust
* To comply with business or legal standards

**Real-Time Use Case:**  
In the healthcare domain, testing is crucial for apps that manage patient records. Any miscalculation in medication dosage due to a bug can be life-threatening.

**4. Error-Failure-Defect**

**Explanation:**

* **Error:** A human mistake (e.g., a developer writes incorrect code).
* **Defect:** A flaw in the software code.
* **Failure:** When the system behaves incorrectly in production due to a defect.

**Real-Time Use Case:**  
If a developer forgets to handle a null value, it causes a defect. When a user triggers that part of the app and it crashes, that is a failure.

**5. Causes of Software Defects**

**Explanation:**

* Misunderstood requirements
* Incorrect logic or design
* Poor coding practices
* Environmental issues (e.g., hardware, OS mismatch)
* Communication gaps

**Real-Time Use Case:**  
A defect may occur in a travel booking app if the backend logic allows booking a flight date that is in the past, due to a misunderstanding between client requirements and developer understanding.

**6. Cost of Software Defects**

**Explanation:**  
The cost of fixing defects increases exponentially the later they are found:

* **Requirement Phase:** Least costly
* **Design/Development Phase:** Moderate
* **Production/Deployment:** Most costly

**Real-Time Use Case:**  
In 2012, Knight Capital lost $440 million in 30 minutes due to an untested software bug deployed into production. The bug triggered unintended trades on the stock market.

**7. What Does Software Testing Reveal?**

Software testing helps uncover various types of problems in a software system before it reaches the end user. Here's a breakdown of what it reveals:

**1. Defects in Code (Functional Bugs)**

**Explanation:**  
These are errors in the logic or implementation that cause the software to behave incorrectly or crash.

**Real-Time Example:**  
In a banking app, when a user transfers ₹1000, but ₹2000 is deducted—this is a **functional bug** caused by a defect in the transaction logic.

**2. Gaps in Requirements**

**Explanation:**  
These are missing or misunderstood features or rules due to poor communication or incomplete documentation during the requirement phase.

**Real-Time Example:**  
An e-commerce client expects a product to be out-of-stock after inventory hits zero, but the website still allows it to be added to the cart. This gap wasn't mentioned in the requirements, so developers didn’t implement it—and testing reveals this issue.

**3. Performance Bottlenecks**

**Explanation:**  
These issues occur when the application slows down, crashes, or behaves unpredictably under load or stress (many users at once).

**Real-Time Example:**  
During a flash sale on a shopping app like Flipkart or Amazon, if the app becomes unresponsive due to high traffic, performance testing can reveal that the database can't handle more than 1000 concurrent requests per second.

**4. UI/UX Issues (User Interface / Experience)**

**Explanation:**  
These are issues related to how the software looks and feels. Poor design, confusing layout, or unresponsive components can make it hard for users to interact with the software.

**Real-Time Example:**  
In a food delivery app, if the “Place Order” button is placed too close to the “Cancel” button, users might accidentally cancel their orders. Usability testing will highlight this poor user experience.

**5. Security Vulnerabilities**

**Explanation:**  
Security testing uncovers loopholes in the software that hackers can exploit to access data or manipulate functionality.

**Real-Time Example:**  
In a social media app (like Instagram or Facebook), if user passwords are stored in plain text instead of encrypted (hashed), a hacker who gains access to the database can steal passwords easily. Security testing would detect this vulnerability.

**8. Importance of Software Testing**

**Explanation:**

* Ensures software reliability and stability
* Increases customer satisfaction
* Saves costs long-term
* Ensures business continuity

**Real-Time Use Case:**  
For a ride-sharing app like Uber, a bug that miscalculates distance or fare can affect thousands of users daily. Regular testing ensures the algorithm works accurately.

**9. Importance of Testing Early in SDLC Phases**

**Explanation:**  
Known as **Shift Left Testing**, it means identifying and resolving issues early in the software development life cycle (SDLC) to reduce cost and time.

**Real-Time Use Case:**  
If test engineers are involved during the **requirements phase**, they can point out ambiguities like "fast login" which needs to be quantified (e.g., within 2 seconds).

**10. Testing and Quality**

**Explanation:**  
Testing ensures quality by validating functionality, performance, and compliance with requirements. However, quality is broader—it includes usability, maintainability, scalability, and more.

**Real-Time Use Case:**  
In the airline industry, the quality of booking software isn't just in its ability to book tickets but also in its ability to scale during sales and promotions, maintain bookings history, and recover gracefully from failures.

Here’s a **complete and in-depth explanation** of the next set of **software testing fundamentals** under **"Quality Perception"**, including **real-time use cases** for each topic to reinforce understanding:

**Quality Perception**

**1. Seven Testing Principles**

**Explanation:**  
These are foundational ideas that guide software testing practices:

1. **Testing shows the presence of defects, not their absence:**  
   You can never prove a system is 100% bug-free.  
   **Use Case:** Even if all tests pass for an ATM software, edge cases might still cause failure in rare scenarios.
2. **Exhaustive testing is impossible:**  
   You can’t test every possible input combination.  
   **Use Case:** A date field that accepts input for 100 years has millions of combinations — you must use representative testing.
3. **Early testing saves time and money:**  
   Bugs caught early are cheaper to fix.  
   **Use Case:** Detecting a requirement flaw in a hospital app in the design phase avoids redoing development.
4. **Defects cluster together:**  
   A few modules contain most bugs (Pareto principle).  
   **Use Case:** Login and payment modules in e-commerce often contain more bugs than the product listing page.
5. **Pesticide paradox:**  
   Repeating the same tests will stop revealing new bugs.  
   **Use Case:** You must update test cases when a new payment gateway is added.
6. **Testing is context dependent:**  
   Testing approach changes based on the software type.  
   **Use Case:** Testing a game app requires different focus (like frame rate and UI) than testing accounting software.
7. **Absence-of-errors is a fallacy:**  
   A bug-free product that doesn't meet business needs is still a failure.  
   **Use Case:** A flawlessly coded school app that lacks attendance reports fails its purpose.

**3. How Testing is Conducted?**

**Explanation:**  
Typically through these steps:

* Requirement analysis
* Test planning
* Test case design
* Test execution
* Defect logging
* Reporting and closure

**Real-Time Use Case:**  
For a food delivery app:

* Test planning includes defining scope (order placement, tracking).
* Execution involves simulating real orders.
* Logging finds issues like incorrect delivery time calculations.

**4. Software Testing – Then (Past)**

**Explanation:**

* Testing was done **after development**
* Mostly manual
* Little focus on user experience or security
* Reactive rather than preventive

**Real-Time Use Case:**  
Older banking systems tested only after complete development, often missing critical edge cases like network failures or user concurrency.

**5. Software Testing – Now (Present)**

**Explanation:**

* Testing begins early (shift-left)
* Agile, DevOps, CI/CD, automation
* Strong focus on usability, performance, security
* Tools like Selenium, JMeter, OWASP used

**Real-Time Use Case:**  
Modern mobile apps like WhatsApp use continuous integration and automation to run hundreds of tests daily with every code change.

**6. Scope of Software Testing**

**Explanation:**  
Scope defines what is tested:

* Functional and non-functional requirements
* Third-party integrations
* Mobile/browser compatibility

**Real-Time Use Case:**  
A ride-hailing app's testing scope includes booking flow, payment, location tracking, app behavior on Android/iOS, and interaction with Google Maps API.

**7. Factors Influencing the Scope of Testing**

**Explanation:**

* Project deadlines
* Budget
* Risk level
* Application complexity
* Regulatory requirements

**Real-Time Use Case:**  
A healthcare app requires **broader scope** due to HIPAA compliance, whereas a marketing website may have **limited scope**.

**8. Risk-Based Testing**

**Explanation:**  
Prioritizing tests based on risk impact and likelihood:

* High-risk areas get more testing
* Low-risk areas may get basic testing

**Real-Time Use Case:**  
In a stock trading platform, risk-based testing focuses heavily on order execution logic and transaction security, rather than on the user profile screen.

**9. Project Risks**

**Explanation:**  
Risks related to **project execution**, such as:

* Tight deadlines
* Budget overrun
* Lack of skilled testers
* Tools not available

**Real-Time Use Case:**  
A project with a short release cycle might miss performance testing due to lack of time — that's a project risk.

**10. Product Risks**

**Explanation:**  
Risks related to **software failure** or **non-performance**, such as:

* Data loss
* Security breach
* Business rule violation

**Real-Time Use Case:**  
In a fintech app, a defect that allows a user to transfer money without balance is a **product risk**.

**11. Need of Independent Testing**

**Explanation:**  
Testing by someone other than the developer ensures:

* Unbiased view
* Fresh perspective
* Better defect detection

**Real-Time Use Case:**  
An airline reservation system is developed in-house but tested by an external QA team to ensure no biased assumptions affect test coverage.

**12. Activities in Fundamental Test Process**

**Explanation:**

1. **Planning & Control**
2. **Analysis & Design**
3. **Implementation & Execution**
4. **Evaluating Exit Criteria**
5. **Test Closure**

**Real-Time Use Case:**  
In a retail POS system:

* **Planning:** Define what's tested (scanning, payment)
* **Design:** Create test cases
* **Execution:** Test it in-store
* **Closure:** Create summary report before go-live

**13. Attributes of a Good Tester**

**Explanation:**

* Attention to detail
* Curiosity
* Communication skills
* Analytical thinking
* Persistence

**Real-Time Use Case:**  
A good tester in a logistics app notices inconsistent delivery time for same-distance orders — an anomaly missed by developers.

**14. Psychology of Testing**

**Explanation:**

* Developers and testers think differently
* Testers must challenge assumptions and explore edge cases
* It requires critical thinking, not just script execution

**Real-Time Use Case:**  
In a hotel booking app, developers assume users always enter check-in before check-out, but a tester tries the reverse and finds a defect.

**15. Code of Ethics for Tester**

**Explanation:**

* Maintain confidentiality
* Report honestly
* Avoid conflicts of interest
* Stay professionally competent

**Real-Time Use Case:**  
If a tester in a healthcare software company finds a major defect, they must report it immediately—even if it delays release.

**16. Limitations of Software Testing**

**Explanation:**

* Cannot guarantee 100% bug-free software
* Limited by time and budget
* Some issues only appear in production
* Test coverage may not be complete

**Real-Time Use Case:**  
Even after extensive testing, a real-world user uploads a 10GB video (never tested) in a video conferencing app, causing the system to crash — highlighting limitations of test coverage.

**🔁 1. SDLC Models**

**Definition:** Structured approach for building software  
**Common Models:**

* **Waterfall:** Linear → Used in government/legacy systems
* **Agile:** Iterative → Preferred by startups

**🔗 2. Testing in SDLC**

* Testing activities are mapped to development stages.
* Early testing = early defect detection = cost saving.

**Example:** Login API tested as it’s developed.

**🧱 3. Test Levels**

| **Level** | **Description** | **Real-Time Example** |
| --- | --- | --- |
| **Component** | Testing individual units | EMI calculator function |
| **Integration** | Testing interfaces | Cart + payment sync |
| **System** | E2E testing | Full hotel booking |
| **Acceptance** | Business validation | Beta testing Uber app |

**⚙️ 4. Test Types**

**✅ Functional Testing**

* Tests "what" the system does  
  **Example:** Password reset functionality

**🚀 Non-Functional Testing**

* Tests "how" the system performs  
  **Examples:**
* **Performance:** Page load < 2s
* **Security:** Login locks after 5 fails
* **Compatibility:** Works on Chrome, Safari

**🔍 5. White-box Testing**

* Internal code logic tested by devs  
  **Example:** Ensure all if-else branches in tax code are covered.

**🔥 6. Smoke & Sanity Testing**

| **Type** | **Purpose** | **Example** |
| --- | --- | --- |
| **Smoke** | Is the build stable? | App loads, major menus work |
| **Sanity** | Did a minor fix work? | Promo code now applies correctly |

**🛠️ 7. Change-Related Testing**

**Types:**

* **Confirmation Testing:** Test the fix itself
* **Regression Testing:** Ensure nothing else broke

**Example:** Fixing checkout error → ensure login, cart, payments still work.

**🧩 8. Maintenance Testing**

* Done after app is released and modified  
  **Triggers:** Bug reports, OS changes, feature requests  
  **Example:** Testing Facebook’s UI after iOS 18 release.

**🔎 9. Impact Analysis**

* Identify what to test after a change  
  **Example:** After payment API change → test refund, transaction history, receipts.

**🧪 10. Test Case Terminology**

| **Term** | **Definition** | **Example** |
| --- | --- | --- |
| **Test Case** | Step-by-step checks | "Login with valid user" |
| **Test Suite** | Group of cases | Login + Logout + Profile |
| **Test Scenario** | User-based flow | "Order food & make payment" |
| **Test Script** | Automation code | Selenium script for search bar |

**📂 11. Test Data**

**Types:**

* Valid
* Invalid
* Boundary
* Random

**Example (Contact Form):**

* Valid: Name = "John", Email = "[john@example.com](mailto:john@example.com)"
* Invalid: Email = "abc"
* Boundary: Name = 50 chars

**🧠 Summary Table: Levels vs Test Types**

| **Level** | **Functional** | **Non-Functional** |
| --- | --- | --- |
| Unit | Logic tests | Execution speed |
| Integration | API calls | Reliability |
| System | Feature flows | Performance, Security |
| Acceptance | Business cases | Usability |

**🎓 Final Thought:**

Testing is not just a phase — it’s a **continuous, integral** part of software development that ensures:

* Quality
* Stability
* Trust
* User satisfaction

Here's a detailed explanation of the topic **"Test Techniques"** — with clear definitions, examples, and real-time use cases for each concept.

**🧪 Test Techniques**

These are systematic methods used to design test cases based on requirements, code, experience, or business logic.

**✅ 1. Categories of Test Techniques**

| **Category** | **Description** |
| --- | --- |
| **Black-box** | Focuses on inputs and outputs, not code |
| **White-box** | Focuses on code structure and logic |
| **Experience-based** | Relies on tester’s knowledge & past issues |

**🎯 2. Choosing Test Techniques**

**Depends on:**

* Type of system (web, embedded, mobile)
* Project timeline and budget
* Risk levels and criticality
* Tester skill set
* Test basis (requirements, code, etc.)

**Example:**  
For safety-critical software (e.g., medical device), white-box techniques are essential. For fast-changing UI apps, exploratory testing works well.

**📚 3. Categories & Characteristics**

| **Category** | **Strengths** | **Limitations** |
| --- | --- | --- |
| Black-box | User-focused, business coverage | May miss code-level issues |
| White-box | Code coverage, logic validation | Needs coding knowledge |
| Experience-based | Fast, intuitive | May lack repeatability |

**🧩 Black-box Test Techniques**

**🔹 4. Equivalence Partitioning (EP)**

**Concept:**  
Divide input into groups (valid/invalid); test one value from each group.

**Example:**  
For an age field (1–100), valid partitions:

* 1–17 (minor)
* 18–60 (adult)
* 61–100 (senior)  
  Invalid: 0, 101, -5

**Use Case:**  
A government portal checks eligibility for voting (18+). Testers select one value from each age partition.

**🔹 5. Boundary Value Analysis (BVA)**

**Concept:**  
Test values at boundaries (min, max, just inside/outside).

**Example:**  
Input = 1–100  
Test values: 0, 1, 2, 99, 100, 101

**Use Case:**  
Online banking app testing withdrawal limits between ₹100–₹10,000.

**🔹 6. Decision Table Testing**

**Concept:**  
Create tables of conditions vs actions to cover combinations.

**Example:**  
Credit card approval:

| **Income >50k** | **Credit Score >700** | **Approve?** |
| --- | --- | --- |
| Yes | Yes | Yes |
| Yes | No | No |
| No | Yes | No |
| No | No | No |

**Use Case:**  
Loan management system testing different eligibility conditions.

**👨‍💻 White-box Test Techniques**

**🔹 9. Statement Testing & Coverage**

**Concept:**  
Verify each line of code is executed at least once.

**Example:**  
In if-else blocks, make sure both parts are tested.

**Use Case:**  
Unit testing in banking apps to ensure interest calculations are not skipped.

**🔹 10. Condition Testing & Coverage**

**Concept:**  
Each decision (true/false) must be evaluated.

**Example:**  
if (age > 60 || income < 20000) → test both true and false paths

**Use Case:**  
Eligibility logic in healthcare systems.

**🔹 11. Value of Statement and Decision Testing**

**Why it matters:**

* Prevents missed code paths
* Detects logic errors
* Essential for **high-risk applications** like avionics or finance

**Use Case:**  
Medical devices require **100% decision coverage** to ensure patient safety.

**🧠 Experience-based Test Techniques**

**🔹 12. Error Guessing**

**Concept:**  
Guess likely problem areas based on experience.

**Example:**  
Testing file upload with .exe extension, long filenames, or emoji in file name.

**Use Case:**  
E-commerce checkout – tester guesses that entering invalid coupon code might crash system.

**🔹 13. Exploratory Testing**

**Concept:**  
Simultaneous learning, test design, and execution. No predefined scripts.

**Use Case:**  
New app with minimal documentation → tester explores to find UI glitches, crashes, inconsistencies.

**🔹 14. Checklist-based Testing**

**Concept:**  
Use checklists for coverage of common defects.

**Example:**  
Login checklist:

* Valid credentials
* Invalid password
* Blank username
* SQL injection attempt
* Password reset flow

**Use Case:**  
Used during quick health-check testing before releases.

**📌 Summary Table**

| **Technique** | **Type** | **Best For** |
| --- | --- | --- |
| Equivalence Partitioning | Black-box | Reducing test data size |
| Boundary Value Analysis | Black-box | Critical numeric ranges |
| Decision Table | Black-box | Business rules, logic combinations |
| Use Case Testing | Black-box | End-to-end workflows |
| Statement Testing | White-box | Code coverage |
| Decision Testing | White-box | Logic correctness |
| Error Guessing | Experience | Common mistakes and edge cases |
| Exploratory Testing | Experience | Early-stage, fast feedback |
| Checklist Testing | Experience | Quick, broad verification |