# 

# Software

**Definition**: Software is a collection of computer programs, procedures, and documentation that perform specific tasks on a computer system. It can be categorized based on its purpose and functionality.

#### **Types of Software**

1. **System Software**
   * **Description**: This type of software manages and operates computer hardware, providing a foundation for application software. It helps manage the system’s resources and ensures communication between hardware and software.
   * **Examples**:
     + **Operating Systems**: Windows, macOS, Linux, which provide a user interface and manage hardware.
     + **Device Drivers**: Printer drivers, sound card drivers, which enable hardware devices to communicate with the OS.
     + **Server Software**: Software like Apache or Nginx, which manages and provides web services.
2. **Programming Software**
   * **Description**: These are tools that developers use to write, test, and debug their code. This type of software helps create new programs and is essential for software development.
   * **Examples**:
     + **Compilers**: Converts high-level programming code into machine code, e.g., GCC for C++.
     + **Debuggers**: Helps identify and fix bugs in the code, e.g., WinDbg for Windows applications.
     + **Interpreters**: Translates code into machine language line-by-line, e.g., Python Interpreter for Python code.
3. **Application Software**
   * **Description**: This type of software is designed for end-users to perform specific tasks, such as browsing, document creation, or entertainment. Application software includes both online (web-based) and offline programs.
   * **Examples**:
     + **Web Applications**: Google Docs, Gmail – accessed via a web browser.
     + **Mobile Applications**: Instagram, WhatsApp – designed specifically for mobile devices.
     + **Desktop Applications**: Microsoft Word, Adobe Photoshop – installed and run on desktop computers.

# Software Testing

**Definition**: Software testing is the process of evaluating and verifying that a software application or product meets the specified requirements and is free of defects. It ensures that the developed software functions as expected and satisfies the needs of the customer.

**Example Scenario**:

* **Customer**: XBank
* **IT Company**: Develops software for XBank
* **Testing Process**: The IT company tests the developed product to ensure it meets XBank's requirements before delivering it. This step ensures the software is ready for use without critical issues.

### **Key Points**

1. **Part of the Software Development Process**
   * Testing is an integral part of the software development lifecycle (SDLC). It involves continuously evaluating the product during development to catch defects early and ensure quality.
2. **Purpose of Testing**:
   * **Detect and Identify Defects**: Testing activities aim to find and document defects or issues in the software.
   * **Ensure Quality**: The ultimate goal of testing is to release a high-quality product to the client that fulfills their requirements and operates reliably in the real world.
3. **Verification and Validation (V&V)**
   * **Verification**: Ensures that the software is designed correctly. It includes activities like reviews, inspections, and walkthroughs. Verification checks if the software meets the design specifications before actual testing begins.
     + *Example*: Reviewing code or design documents to ensure they follow guidelines and requirements.
   * **Validation**: Ensures that the software is built correctly and meets the customer's needs. It involves actual execution of the software to check for expected behavior.
     + *Example*: Running test cases to confirm that the login feature in a banking app works as specified.

### **Objective of Software Testing**

* The main objective is to deliver a **quality product** that meets client expectations, is free from critical bugs, and operates reliably. Quality is assessed based on factors like performance, security, and usability.

# Software Quality

**Definition**: Software quality refers to the degree to which a software product meets specified requirements, is free from defects, and fulfills customer expectations. High-quality software is reliable, efficient, and satisfies the intended use.

#### **Key Parameters of Software Quality**

1. **Bug-Free**
   * **Description**: A high-quality software product should have minimal to no defects or issues that impact its functionality. Achieving a bug-free state means the product has undergone thorough testing and defect correction.
   * **Example**: A banking application that processes transactions accurately without unexpected crashes or errors.
2. **Delivered On Time**
   * **Description**: Quality also means meeting the agreed timeline. Timely delivery is essential to satisfy customer needs and maintain project milestones, as delayed software can lead to missed market opportunities and increased costs.
   * **Example**: If a retail company launches a holiday shopping app, it must be delivered before the shopping season to be useful.
3. **Within Budget**
   * **Description**: Staying within budget is another aspect of software quality, as cost overruns can negatively impact client satisfaction and profitability. Efficient use of resources to meet requirements within the allocated budget is key.
   * **Example**: A project budgeted at $50,000 that meets quality standards without exceeding this cost shows good budget adherence.

# Project vs. Product

1. **Project**
   * **Definition**: A project in software development is a temporary endeavor undertaken to create a unique software application for a specific customer or organization, often with a set timeline and specific requirements.
   * **Characteristics**:
     + Developed for a **specific customer** or client.
     + Has a clear start and end date.
     + Goals and requirements are usually defined by the customer.
   * **Example**: A custom banking app developed specifically for XBank, with features tailored to XBank’s internal operations and customer requirements.
2. **Product**
   * **Definition**: A software product is a general-purpose application developed for a broader audience. It is designed to meet the needs of multiple customers or users, often with standardized features and functionality.
   * **Characteristics**:
     + Created for **multiple customers** or the general market.
     + Typically has ongoing development, updates, and maintenance.
     + Features are designed based on market needs and are not customized for individual users.
   * **Example**: Microsoft Office Suite, which is developed for a wide range of users across various industries, or an antivirus program that caters to any user who needs protection.

# Verification (Static Testing)

* **Definition**: Verification ensures that the software product is being built correctly by reviewing documents and code without executing the code.
* **Objective**: Check that we are building the right product according to specifications.
* **Focus**: Documents and requirements.
* **Examples**:
  + Reviewing a requirements document to ensure completeness.

# Inspecting a codebase to verify adherence to coding standards.

### Peer Review

**Definition**: Peer review is a process where team members review documents or code created by others to identify issues and improve quality. It varies in formality and preparation.

### **Review Types in Software Testing**

1. **Review**
   * **Purpose**: Conducted on project documents to ensure they meet standards and requirements, reviews are essential for catching issues early.
   * **Documents Reviewed**: Requirements, design documents, code, test plans, and test cases.
   * **Example**: Reviewing a requirements document to confirm it aligns with client specifications.
2. **Walkthrough**
   * **Description**: An informal review where the author reads through the document or code with the team. Attendees do not need prior preparation.
   * **Purpose**: Aimed at knowledge sharing and gathering early feedback.
   * **Example**: A developer explaining the architecture of a new feature to the team to clarify understanding and get feedback.
3. **Inspection**
   * **Description**: A formal, structured review with specific roles (e.g., reader, writer, moderator) that requires prior preparation.
   * **Purpose**: Systematic defect identification to ensure high-quality standards.
   * **Example**: Reviewing a final design document for a critical system before coding, ensuring it aligns with standards and requirements.

# 

# Validation (Dynamic Testing)

* **Definition**: Validation ensures that the final product functions as intended by executing the software and observing its behavior.
* **Objective**: Check that we have built the product correctly according to user expectations.
* **Focus**: The software in action.
* **Example**: Running functional tests on a web application to verify that all features perform as expected.

### **Validation Strategies (Dynamic Testing)**

1. **Black Box Testing**
   * **Description**: Testing the software’s functionality without knowing the internal code structure. It focuses on providing inputs and observing outputs.
   * **Focus**: Behavioral testing.
   * **Techniques**:
     + **Equivalence Partitioning**: Dividing input data into valid and invalid groups to reduce test cases.
     + **Boundary Value Analysis**: Testing at the edge of input ranges.
     + **Decision Table Testing**: Using tables to test combinations of inputs.
   * **Example**: Testing a login page by entering valid and invalid usernames and passwords without knowing how the code handles them.
2. **White Box Testing**
   * **Description**: Testing that requires knowledge of the internal code and is generally done by developers. It aims to check the internal structure, flows, and coverage.
   * **Focus**: Structural or transparent testing.
   * **Techniques**:
     + **Statement Coverage**: Ensuring each line of code is executed at least once.
     + **Branch/Decision Coverage**: Testing all possible outcomes of each decision point.
     + **Condition Coverage**: Testing all individual conditions within decision points.
   * **Example**: Unit testing where developers test each function to cover various code paths and conditions.

### 

### **Static vs. Dynamic Testing**

1. **Static Testing**:
   * **Definition**: Examining documents or code without execution, focusing on verifying correctness and completeness.
   * **Methods**:
     + Review, Walkthrough, Inspection.
   * **Example**: Reviewing a test plan document for thoroughness before starting dynamic testing.
2. **Dynamic Testing**:
   * **Definition**: Testing the software by executing it to validate its behavior and performance.
   * **Example**: Running functional or system tests on an application to ensure it works as expected.

# Software Development Life Cycle (SDLC)

**Definition**: The Software Development Life Cycle (SDLC) is a process used by the software industry to design, develop, and test software applications systematically. It serves as a structured framework to ensure that software products meet quality standards and customer requirements.

**3 P’s of SDLC**:

* **People** follow a **Process** to create a **Product**.

### **Key Phases of SDLC**

1. **Requirement Analysis**
   * Gathering and analyzing requirements from the customer to understand what the software needs to achieve.
   * **Example**: For a banking app, requirements might include login, funds transfer, and transaction history features.
2. **Design**
   * Creating a blueprint for the software, including architecture and design specifications.
   * **Example**: Designing the database schema and UI/UX layout for the banking app.
3. **Development**
   * Writing code based on the design specifications to build the application.
   * **Example**: Coding the login and transaction processing functions for the app.
4. **Testing**
   * Verifying that the software works as expected and is free of defects.
   * **Example**: Running tests to check if the login feature accepts only valid credentials.
5. **Maintenance**
   * Ongoing support and updates after the software is released to fix bugs or add new features.
   * **Example**: Updating the banking app to include new payment options.

### 

### 

### 

### 

### **SDLC Models**

1. **Waterfall Model**
   * **Description**: A linear and sequential model where each phase is completed before moving to the next. It involves thorough documentation at each stage.
   * **Advantages**:
     + High-quality output with comprehensive documentation.
     + Fewer bugs due to clear planning and execution.
   * **Disadvantages**:
     + Changes in requirements are difficult to incorporate once the process starts.
     + Best suited for small, well-defined projects where requirements are unlikely to change.
   * **Example**: Developing an accounting system with fixed requirements for a small business.
2. **Spiral Model**
   * **Description**: Combines iterative and waterfall models, focusing on risk analysis and multiple cycles (iterations) of development.
   * **Key Phases**:
     + **Planning**: Identifying requirements.
     + **Risk Analysis**: Identifying and addressing potential risks.
     + **Engineering**: Coding and testing.
     + **Evaluation**: Assessing the software after each iteration.
   * **Advantages**:
     + New versions of the software are released in each cycle.
     + Testing is done at each iteration, making it easier to detect defects early.
   * **Example**: A complex e-commerce platform with changing requirements over time.
3. **Prototype Model**
   * **Description**: A preliminary model (prototype) of the software is created to gather customer feedback. This model is refined based on customer input before final development.
   * **Steps**:
     + Gathering initial requirements.
     + Designing and developing a prototype.
     + Testing and receiving customer feedback.
   * **Advantages**:
     + Helps in understanding customer requirements more clearly.
     + Useful when requirements are not well defined.
   * **Example**: Developing a prototype for a mobile app to show the layout and navigation before finalizing the design.
4. **V-Model (Verification and Validation Model)**
   * **Description**: A sequential model that emphasizes verification (static testing) and validation (dynamic testing). Each development phase has a corresponding testing phase.
   * **Stages**:
     + **BRS/CRS/URS** (Business, Customer, and User Requirement Specifications): Verification with User Acceptance Testing (UAT) to ensure it meets business needs.
     + **SRS (System Requirement Specification)**: Verification with System Testing to check the functionality.
     + **Design Specification**: Verification with Integration Testing to test module interactions.
     + **Coding**: Verification with Unit Testing to check each individual component.
   * **Advantages**:
     + Emphasizes testing at each stage, leading to fewer defects.
   * **Example**: Developing a government system where strict compliance and systematic validation are crucial.

# QA vs. QC vs. QE

1. **Quality Assurance (QA)**
   * **Definition**: A process-oriented approach focused on improving and ensuring the quality of the development process. QA aims to prevent defects by setting standards and guidelines for building a quality product.
   * **Characteristics**:
     + **Process-Oriented**: Focuses on defining and improving processes.
     + **Goal**: Ensures the quality process is followed to prevent defects.
     + **Entire Lifecycle**: QA activities span the entire SDLC.
   * **Example**: Creating and enforcing a code review process to ensure coding standards are met, reducing the likelihood of bugs.
2. **Quality Control (QC)**
   * **Definition**: A product-oriented approach that involves actual testing of the software to detect and correct defects. QC verifies that the product meets quality standards after development.
   * **Characteristics**:
     + **Product-Oriented**: Focuses on identifying and fixing defects in the product.
     + **Goal**: Detects defects in the final product to ensure it meets customer expectations.
     + **Part of Testing in SDLC**: QC is usually performed in the testing phase.
   * **Example**: Running test cases on a new software release to find and fix bugs before the product goes live.
3. **Quality Engineering (QE)**
   * **Definition**: A people-oriented approach that combines QA and QC, focusing on automating quality processes. QE uses automation to streamline testing and improve quality throughout the lifecycle.
   * **Characteristics**:
     + **Automation-Oriented**: Uses automation tools to improve testing and quality processes.
     + **Goal**: Builds quality by automating testing processes, enabling continuous integration and delivery.
     + **Continuous Lifecycle**: QE activities are integrated throughout the development process.
   * **Example**: Using automated test scripts to run regression tests in CI/CD pipelines, ensuring quick feedback on code quality with each new code change.

### 

### 

### **Key Differences Summary**

| **Aspect** | **Quality Assurance (QA)** | **Quality Control (QC)** | **Quality Engineering (QE)** |
| --- | --- | --- | --- |
| **Focus** | Process-oriented | Product-oriented | Automation and people-oriented |
| **Objective** | Prevent defects | Detect and fix defects | Automate quality processes |
| **Stage in SDLC** | Entire lifecycle | Testing phase | Continuous lifecycle |
| **Example Activity** | Setting coding standards | Executing test cases | Implementing automated testing |

# Levels of Testing

Testing levels ensure that software is evaluated at different stages of development, from individual components to the complete system, to verify functionality and quality.

#### **1. Unit Testing**

* **Definition**: Low-level testing of individual code units or functions to ensure they work correctly.
* **Performed By**: Developers
* **Objective**: Validate that each unit performs as expected.
* **Example**: Testing a function add() to verify it returns the correct sum.
* **Techniques**:
  + **Basic Path**: Ensures every line of code is executed at least once.
  + **Control Structure Testing**: Covers conditional branches, loops, and decision points.
  + **Conditional Coverage**: Ensures each condition in the code is tested.
  + **Loop Coverage**: Tests loops within the code for all possible iterations.

#### **2. Integration Testing**

* **Definition**: Testing the interfaces and interactions between multiple modules, focusing on data flow and communication.
* **Performed By**: Developers (often includes collaboration with testers)
* **Objective**: Validate that modules work together correctly when integrated.
* **Example**: Testing data flow between add() and subtract() functions within a math module.
* **Types of Integration Testing**:
  1. **Incremental Integration Testing**: Modules are integrated and tested incrementally, ensuring each new module works with the previous modules.
     + **Top-Down Approach**: Higher-level modules are tested first, moving down the hierarchy. Lower-level modules are tested by creating *stubs* (dummy modules).
     + **Bottom-Up Approach**: Lower-level modules are tested first, moving up the hierarchy. Higher-level modules are tested by creating *drivers* (dummy calling programs).
     + **Sandwich/Hybrid Approach**: Combines both top-down and bottom-up approaches.
  2. **Non-Incremental (Big Bang) Testing**: All modules are integrated simultaneously, and then the data flow between them is tested as a whole.

#### **3. System Testing**

* **Definition**: High-level testing of the entire integrated system to evaluate its compliance with specified requirements.
* **Performed By**: Testers
* **Objective**: Verify the software as a whole, ensuring that it meets the business and functional requirements.
* **Example**: Testing a banking application to validate all features (like transfer, withdrawal, balance inquiry) function together as expected.
* **Types/Goals of System Testing**:
  + **Usability Testing**: Ensures the system is user-friendly and intuitive.
  + **Performance Testing**: Measures system performance under various conditions.
    - *Load Testing*: Tests the system under expected user load.
    - *Stress Testing*: Evaluates system behavior under extreme conditions.
    - *Volume Testing*: Checks system performance with large volumes of data.
  + **Security Testing**: Assesses the system’s security mechanisms against potential threats.
  + **Configuration Testing**: Validates the system's functionality across different configurations.
  + **Compatibility Testing**: Ensures compatibility with different operating systems, browsers, etc.
  + **Installability Testing**: Confirms that the software installs and uninstalls properly.
  + **Recovery Testing**: Checks the system's ability to recover from crashes or failures.
  + **Availability Testing**: Tests if the system is available and accessible as expected.

#### **4. User Acceptance Testing (UAT)**

* **Definition**: Final testing phase where the software is tested in a real-world environment to verify if it meets customer requirements.
* **Performed By**: End-users or clients
* **Objective**: Ensure the software satisfies business needs and is ready for production deployment.
* **Example**: Testing an e-commerce website’s purchase workflow by clients to ensure it meets the necessary requirements before release.

### 

### 

### 

### **Additional Concepts in Testing**

* **Component Testing**: Testing multiple units together, such as testing add(), subtract(), and divide() functions as a group within a math module.
* **Module Testing**: Testing a collection of components, such as multiple math functions within a module, to ensure they work as a whole.
* **Stubs**: Used in top-down integration testing as temporary substitutes for lower-level modules that aren’t yet developed.
* **Drivers**: Used in bottom-up integration testing as temporary substitutes for higher-level modules that call the lower modules.

# Types of Testing

1. **Alpha (α) Testing**
   * **Definition**: Testing conducted within the organization by internal staff to detect bugs before releasing the software to the public.
   * **Example**: A software company tests a new version of an app in-house with real scenarios, ensuring it functions correctly before releasing a beta version to select users.
2. **Beta (β) Testing**
   * **Definition**: Testing done by a select group of external users in a real environment, prior to full release. It helps gather feedback and identify bugs missed during internal testing.
   * **Example**: A beta version of a mobile app is released to users worldwide, who test it under various conditions and provide feedback.
3. **Final Candidate Testing**
   * **Definition**: The last round of testing done just before the official release to verify that the product is ready for market launch.
   * **Example**: Testing an app’s final build to ensure all major bugs have been fixed and it meets performance standards.

### **Specific Types of Tests**

1. **Retesting (Confirmation Testing)**
   * **Definition**: Testing to confirm that a defect has been fixed successfully.
   * **Example**: If a bug in the login feature was reported, retesting would involve checking that the bug no longer exists after the fix.
2. **Regression Testing**
   * **Definition**: Re-executing previously passed test cases to ensure recent code changes have not affected existing functionality.
   * **Example**: After modifying the login feature, regression testing involves re-running previous test cases (e.g., cancel button functionality) to ensure nothing else is broken.
   * T**esting a Login Feature Across Builds**
3. **Initial Test Case** (Build A)
   * Test login with valid and invalid credentials (TC10).
4. **Updated Test Case** (Build B)
   * A new feature (e.g., password recovery) is added.
   * Test login functionality plus additional test cases (TC5) related to new features, making a total of TC15.
5. **Regression Testing Scenario**
   * After changes in Build B, confirm that the "Cancel" button in the login feature behaves as expected and has not been affected by the recent updates.
6. **Progressive Testing**
   * **Definition**: Testing done when new code changes do not affect existing features, typically focusing on newly added features.
   * **Example**: When a new password recovery feature is added, progressive testing checks only this feature without retesting other parts of the application.
7. **Smoke Testing (Build Verification Testing)**
   * **Definition**: A quick check of basic, critical functionalities to verify that the build is stable enough for further testing.
   * **Example**: Upon receiving a new build, testers check if the app launches, if login works, and if main features are accessible.
8. **Sanity Testing**
   * **Definition**: Testing specific areas of an application to confirm that a particular function or bug fix works as expected.
   * **Example**: After fixing a bug in the checkout process, sanity testing verifies only the checkout process without going through all test cases.

### **Special Types of Testing**

1. **Internationalization Testing**
   * **Definition**: Verifying that the application functions across different languages, regions, and formats.
   * **Example**: Testing a website to ensure that currency, date, and time formats change appropriately for users in the U.S., Europe, and Asia.
2. **Mutation Testing**
   * **Definition**: Modifying the program’s source code in small ways (mutants) to check if the existing test cases can detect these changes. It ensures that test cases are robust.
   * **Example**: Changing an operator in a function (e.g., replacing + with -) to see if test cases can catch the difference.

# System Testing

**Definition**: System testing is the process of testing the complete and integrated software application to ensure it meets the specified requirements. It verifies the overall functionality of the application and checks if it behaves as expected in a real-world environment.

#### **1. Types of System Testing**

1. **Graphical User Interface (GUI) Testing**
   * **Definition**: Testing the visual elements of the application to ensure they are displayed correctly and function as intended.
   * **Example**: Verifying that buttons, menus, and checkboxes are present, clickable, and respond correctly to user interactions (e.g., a "Submit" button should lead to the expected action).
2. **Usability Testing**
   * **Definition**: Assessing the application for user-friendliness and ease of use. This involves ensuring that users can navigate the application without confusion and that help features are accessible.
   * **Example**: Testing an e-commerce website to ensure that users can easily find products and access help documentation when needed.

#### **2. User Acceptance Testing (UAT)**

* **Definition**: A final round of testing performed by end users to validate that the system meets their requirements and is ready for production.
* **Example**: A client tests the application in a real-world scenario to ensure that it meets their business needs and functionalities before going live.

#### **3. Testing Approaches**

* **Alpha Testing**: Conducted in-house by the development team to identify bugs before releasing the software to external users.
* **Beta Testing**: Released to a limited number of external users to gather feedback and identify any remaining issues before the official launch.

# Functional Testing

**Definition**: Testing the application against functional requirements to ensure it behaves as expected.

* **Object Properties Testing**: Checking properties of UI elements (e.g., ensuring a button is disabled when conditions aren't met).
* **Database Testing**: Verifying that the user interface communicates correctly with the database, including data manipulation language (DML) operations, and ensuring data integrity and relationships between tables (e.g., checking foreign key constraints).
* **Error Handling Testing**: Validating that the application displays appropriate error messages when users perform invalid actions (e.g., trying to log in with incorrect credentials).
* **Calculations/Manipulation Testing**
  + **Definition**: Verifying that all calculations performed by the application are correct and yield expected results.
  + **Example**: Testing a financial application to ensure that it correctly calculates taxes, discounts, and total amounts based on user inputs.
* **Links Testing**
  + **Definition**: Checking that all hyperlinks within the application are valid and lead to the correct destinations.
  + **Example**: Testing a website to ensure that all navigation links redirect to the intended pages without errors.
* **Cookies & Sessions Testing**
  + **Definition**: Testing how the application handles cookies and session data to ensure user data is stored correctly and sessions are managed securely.
  + **Example**: Verifying that user login sessions expire after a period of inactivity and that cookies are set appropriately when a user logs in.

# Non-Functional Testing

**Definition**: Non-functional testing focuses on the performance, usability, reliability, and other aspects of a software application that are not related to specific functionalities. It assesses how the system performs under various conditions and meets certain standards.

#### **Types of Non-Functional Testing**

1. **Performance Testing**
   * **Definition**: Evaluates the speed, responsiveness, and stability of the application under a particular workload.
   * **Subtypes**:
     + **Load Testing**: Determines how the system behaves under expected user load.
       - **Example**: Testing a web application with 1,000 simultaneous users to see how it handles normal traffic.
     + **Stress Testing**: Tests the application beyond its normal operational capacity to identify breaking points.
       - **Example**: Gradually increasing the number of users from 1,000 to 10,000 and observing at which point the application fails or slows down significantly.
     + **Volume Testing**: Checks the application's performance with a large volume of data to ensure it can handle data processing requirements.
       - **Example**: Loading a database with millions of records to assess query performance and system behavior.
2. **Security Testing**
   * **Definition**: Assesses the application’s ability to protect data and maintain functionality as intended, focusing on vulnerabilities and potential threats.
   * **Key Areas**:
     + **Authentication**: Verifies that users are who they say they are.
       - **Example**: Testing login processes to ensure that valid users can access the system and invalid users are denied access.
     + **Authorization**: Ensures users have permission to access specific resources or functions.
       - **Example**: Testing whether a regular user can access admin functionalities, which should be restricted.
     + **Access Control**: Confirms that users have the appropriate level of access based on their roles.
3. **Recovery Testing**
   * **Definition**: Validates the system's ability to recover from failures or crashes and resume normal operations.
   * **Example**: Simulating a server crash and then checking whether the application can restore data and functionality effectively after a reboot.
4. **Compatibility Testing**
   * **Definition**: Ensures that the application works correctly across different environments, including various operating systems, browsers, and hardware configurations.
   * **Subtypes**:
     + **Forward Compatibility Testing**: Ensures the application works with future versions of software and hardware.
       - **Example**: Testing an application on the latest operating system version to see if it functions correctly.
     + **Backward Compatibility Testing**: Verifies that the application works with older versions of software and hardware.
       - **Example**: Ensuring that an app built for the latest version of a browser still functions correctly on older versions.
5. **Configuration Testing**
   * **Definition**: Checks the application’s behavior and performance in various configurations and environments.
   * **Example**: Testing an application on different hardware setups (e.g., varying RAM, CPU) to see how performance varies.
6. **Sanitation (Garbage Testing)**
   * **Definition**: Involves removing unnecessary features or functionalities that are beyond the requirements to ensure the application is streamlined.
   * **Example**: Analyzing an application for redundant features and removing them to enhance performance and reduce complexity.

# Software Testing Terminologies

1. **Regression Testing**
   * **Definition**: A type of testing to confirm that recent changes to the code (such as enhancements or bug fixes) have not adversely affected existing functionalities.
   * **Example**: After modifying a login feature, regression testing would involve re-running existing tests to ensure that the login still works as intended along with other unaffected features.
2. **Unit Testing**
   * **Definition**: Testing individual components or modules of a software application in isolation to verify that each part functions correctly.
   * **Example**: Testing a function that calculates the total price in an e-commerce application to ensure it returns the correct value when given different inputs.
3. **Integration Testing**
   * **Definition**: Testing the interaction between integrated components or systems to ensure they work together as expected.
   * **Example**: Verifying that the payment processing module interacts correctly with the inventory management system after an order is placed.
4. **Retesting**
   * **Definition**: The process of testing a specific defect after it has been fixed to ensure that the issue is resolved.
   * **Example**: If a bug was identified in the checkout process, retesting would involve performing the checkout again to confirm that the bug no longer exists.
5. **Smoke Testing**
   * **Definition**: A preliminary test to check the basic functionality of an application after a new build. It acts as a health check to ensure the software is stable enough for further testing.
   * **Example**: After deploying a new version of a web application, smoke testing might include checking if the home page loads and if users can log in.
6. **Sanity Testing**
   * **Definition**: A subset of regression testing conducted to verify that a particular function or bug fix works as intended after changes have been made.
   * **Example**: After a bug fix in the user registration process, sanity testing would focus on that specific functionality to ensure it now operates correctly.
7. **Exploratory Testing**
   * **Definition**: An informal testing approach where testers explore the application without predefined test cases, aiming to identify defects by understanding the application and its functionalities.
   * **Example**: A tester navigates through a newly released feature, trying different workflows to uncover unexpected behavior.
8. **Ad-hoc Testing**
   * **Definition**: A form of testing that is unstructured and informal, where tests are conducted randomly without any formal test plan or documentation.
   * **Example**: A tester might randomly click through various parts of an application to see if any errors occur, without following a specific test case.
9. **Monkey Testing**
   * **Definition**: A type of random testing where the tester inputs random data into the application without any knowledge of its structure or logic.
   * **Example**: A tester clicks buttons and fills out forms with random values to see if the application can handle unexpected input without crashing.
10. **Positive Testing**
    * **Definition**: Testing the application with valid inputs to ensure it behaves as expected.
    * **Example**: Submitting a valid email address during user registration to confirm the system accepts it.
11. **Negative Testing**
    * **Definition**: Testing the application with invalid inputs to ensure it gracefully handles errors and does not crash.
    * **Example**: Entering an incorrect email format (e.g., "user@@example.com") to verify that the system displays an appropriate error message.
12. **End-to-End Testing**
    * **Definition**: Testing the entire application flow from start to finish to ensure that all integrated components function together correctly, including data integrity among all modules.
    * **Example**: Testing the complete user journey in an online shopping application, from browsing products to placing an order and confirming payment.
13. **Globalization Testing**
    * **Definition**: Testing the application’s ability to function in various cultural and local environments, ensuring it can adapt to different languages and formats.
    * **Example**: Verifying that a software application correctly displays dates, currencies, and addresses according to different regional standards.
14. **Localization Testing**
    * **Definition**: Testing the application to ensure it is correctly adapted for a specific geographical area, including language translation and cultural appropriateness.
    * **Example**: Ensuring that a mobile application provides accurate translations of text and culturally relevant content for users in Japan.

# Test Design Techniques

Test design techniques are essential for preparing effective test data and ensuring comprehensive coverage of various scenarios in software testing. Here are some key techniques:

1. **Equivalence Class Partitioning (ECP)**
   * **Definition**: This technique involves dividing input data into equivalent classes or partitions, where each class is expected to be treated the same by the software. It helps to reduce the number of test cases by selecting representative values from each partition.
   * **Example**: If an application accepts ages between 18 and 35, the equivalence classes might be:
     + Invalid classes:
       - Class 1: Ages < 18 (e.g., 17, 0)
       - Class 2: Ages > 35 (e.g., 36, 500)
     + Valid class:
       - Class 3: Ages 18 to 35 (e.g., 18, 19, 20, ..., 35)
   * **Test Cases**: You can test one value from each class (e.g., 17, 18, 35, 36).
2. **Boundary Value Analysis (BVA)**
   * **Definition**: This technique focuses on testing the boundaries between equivalence classes. Since errors often occur at the edges, boundary values are critical for validation.
   * **Example**: For the age input of 18 to 35, the boundary values to test would include:
     + Below minimum: 17
     + Minimum: 18
     + Just above minimum: 19
     + Maximum: 35
     + Just above maximum: 36
     + Above maximum: 37
   * **Test Cases**: Test inputs 17, 18, 19, 35, 36, and 37.
3. **Decision Table Testing**
   * **Definition**: This technique is used to model complex business logic by creating a table that maps combinations of inputs (conditions) to their corresponding outputs (actions). It helps to ensure that all possible scenarios are covered.
   * **Example**: A simple decision table for a loan approval system might include:

| **Condition 1 (Credit Score)** | **Condition 2 (Income)** | **Action (Loan Approval)** |
| --- | --- | --- |
| >700 | >50,000 | Approve |
| >700 | <=50,000 | Review |
| <=700 | >50,000 | Review |
| <=700 | <=50,000 | Deny |

1. **State Transition Testing**
   * **Definition**: This technique is used to test the behavior of an application as it transitions from one state to another in response to events or inputs. It helps in understanding how the application reacts to different conditions.
   * **Example**: For a login system, states might include:
     + State 1: User Not Logged In
     + State 2: User Logged In
     + Events might include: Entering Credentials, Validating Credentials, Logging Out.
   * **Test Cases**: Test transitions such as:
     + From "Not Logged In" to "Logged In" when valid credentials are provided.
     + From "Logged In" to "Not Logged In" when the user logs out.
2. **Error Guessing**
   * **Definition**: This informal technique relies on the tester’s experience and intuition to identify potential error-prone areas in the application. It involves guessing inputs that are likely to produce errors.
   * **Example**: If a web form accepts only numeric input for a phone number, a tester might input letters, special characters, or excessively long strings to see how the application handles invalid data.
   * **Test Cases**: Test cases could include inputs like "abc123", "!@#$%", or a phone number longer than the expected length.