Testing throughout the software lifecycle

JAN 2026

Contents

- Software Development Lifecycle Model
- Software Testing Levels
- Types of Testing
- Maintenance Testing
- Quiz

Software Development Lifecycle Model

Introduction

- **Software** is more than **just a program code**. A program is an executable code, which serves some **computational purpose.**
- **Software** is considered to be **collection of executable programming code, associated libraries and documentations**. Software, when made for a specific requirement is called **software product.**
- **Engineering** on the other hand, is all about **developing products, using well-defined, scientific principles and methods**.

Importance of Software Engineering

- **Improves Quality:** Software engineering ensures the development of r**eliable, error-free, and high-performance software.**
- **Reduces Cost and Time:** It helps minimize development time and cost through **proper planning and efficient** processes.
- **Handles Complexity**: It breaks down **complex systems into manageable parts,** making development easier and more organized.
- **Enhances Maintainability:** Well-engineered software is easier to **update, fix, and improve** in the future.
- **Boosts Customer Satisfaction:** It **delivers software** that meets user needs and performs as expected.

Characteristics of Good Software

The factors are divided into **three** categories:

1. Operational Phase : The phase where software is actively used in the real environment.

Characteristics:

- **Delivers core functionality.**
- **Handles real-time user input/output.**
- **Must be stable and reliable.**

2. Transitional Phase : The phase of moving software from development to production.

Characteristics:

- **Includes installation & deployment.**
- **Involves user training and documentation.**
- **Requires environment setup and testing.**

Characteristics of Good Software

3. Maintenance Phase : The phase after deployment for updates and issue resolution.

Characteristics:

- **Fixes bugs and errors (corrective).**
- **Adapts to new environments (adaptive).**
- **Adds improvements (perfective) or prevents issues (preventive).**

SDLC Model

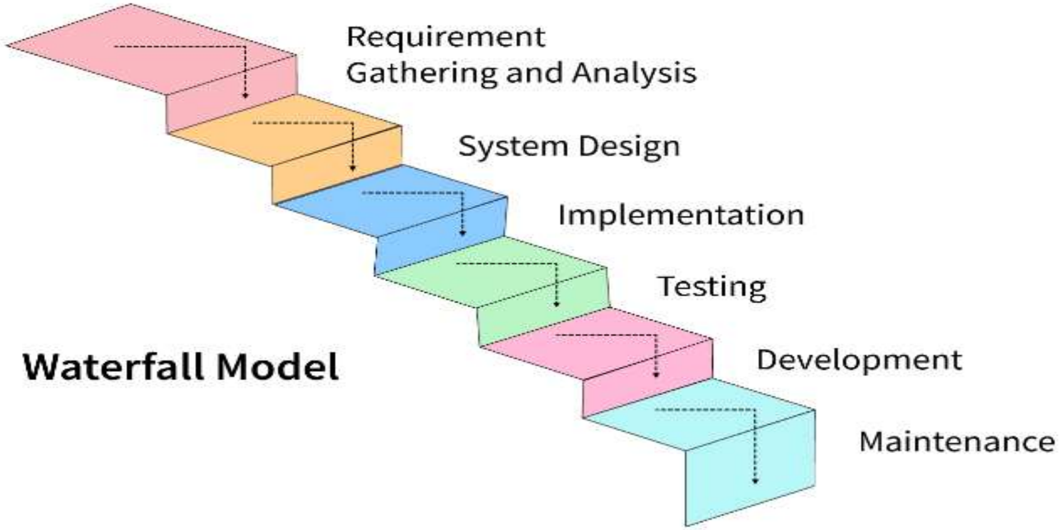
- Various SDLC models have been developed to manage **software projects efficiently**, each suited to different **project needs** and team dynamics. Below are **widely used SDLC models,** each offering a **unique approach to software development,**

  1. Waterfall Model
  2. Iterative Model
  3. V Model
  4. Spiral Model
  5. Agile Model

Types of SDLC Models

Waterfall Model

- The Waterfall Model is one of the **earliest and most traditional software development life cycle** (SDLC) models. It is **a linear and sequential approach,** where each phase must be **completed** before moving on to the next. The process **flows steadily downwards**, like a waterfall, through several distinct phases.



Waterfall Model : Advantages and Disadvantages

Advantages

- Easy to understand, easy to use
- Provide structure to inexperienced staff
- Milestones are well understood
- Sets requirements stability
- Good for management control (plan, staff, track)
- Works well when quality is more important than cost or schedule

Disadvantages

- All requirements must be known upfront
- Deliverables created for each phase are considered frozen – inhibits flexibility
- Can give a false impression of progress
- Does not reflect problem-solving nature of software development – iterations of phases
- Integration is one big bang at the end
- Little opportunity for customer to preview the system (until it may be too late)

When to use waterfall model?

- Requirements are very well known
- Product definition is stable
- Technology is understood
- New version of an existing product
- Porting an existing product to a new platform

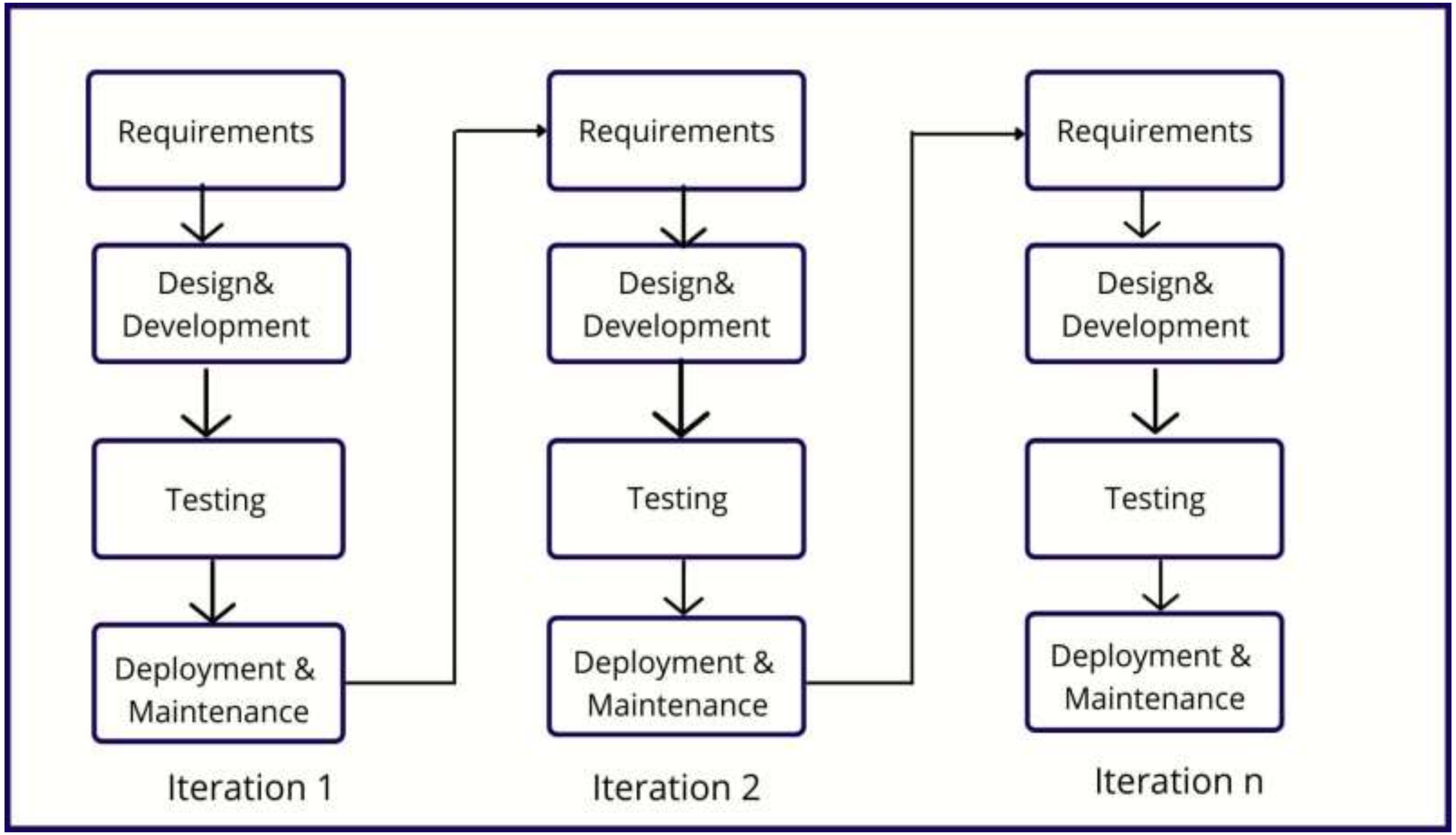
Types of SDLC Models

Iterative Model

- The **iterative process model** is the implementation of the software development life cycle in which the **initial development is started based on the initial requirements** and **more features** are added to the base software product with the **ongoing iterations until the final system is created.**
- Iterative development, in a nutshell, is a method of **breaking down the software development of a massive program into smaller components**.

Types of SDLC Models

Iterative Model



Iterative Model : Advantages and Disadvantages

Advantages

- Produces a working software much quickly and early during the SDLC.
- Very flexible. As new functionality can be added to it at any time of development.
- is less costly to change requirements as compared to the other process models.
- The end-user or the stakeholders can give their feedback quickly.
- The errors and bugs in the system can be identified early.
- Takes smaller development teams as compared to other process models.

Disadvantages

- Problems pertaining to the system architecture can come up because all the requirements are not gathered upfront.
- It is not a good choice for small projects.
- More resource-intensive than waterfall model.
- Risk analysis requires highly qualified specialists to check the risks in our system.
- The whole process is difficult to manage.

When to use Iterative model?

- The iterative model really starts to shine when its in the **hands of a smaller, more agile team.**
- The iterative model is really beneficial as it **can accommodate changes.**
- **Early work product is needed.**

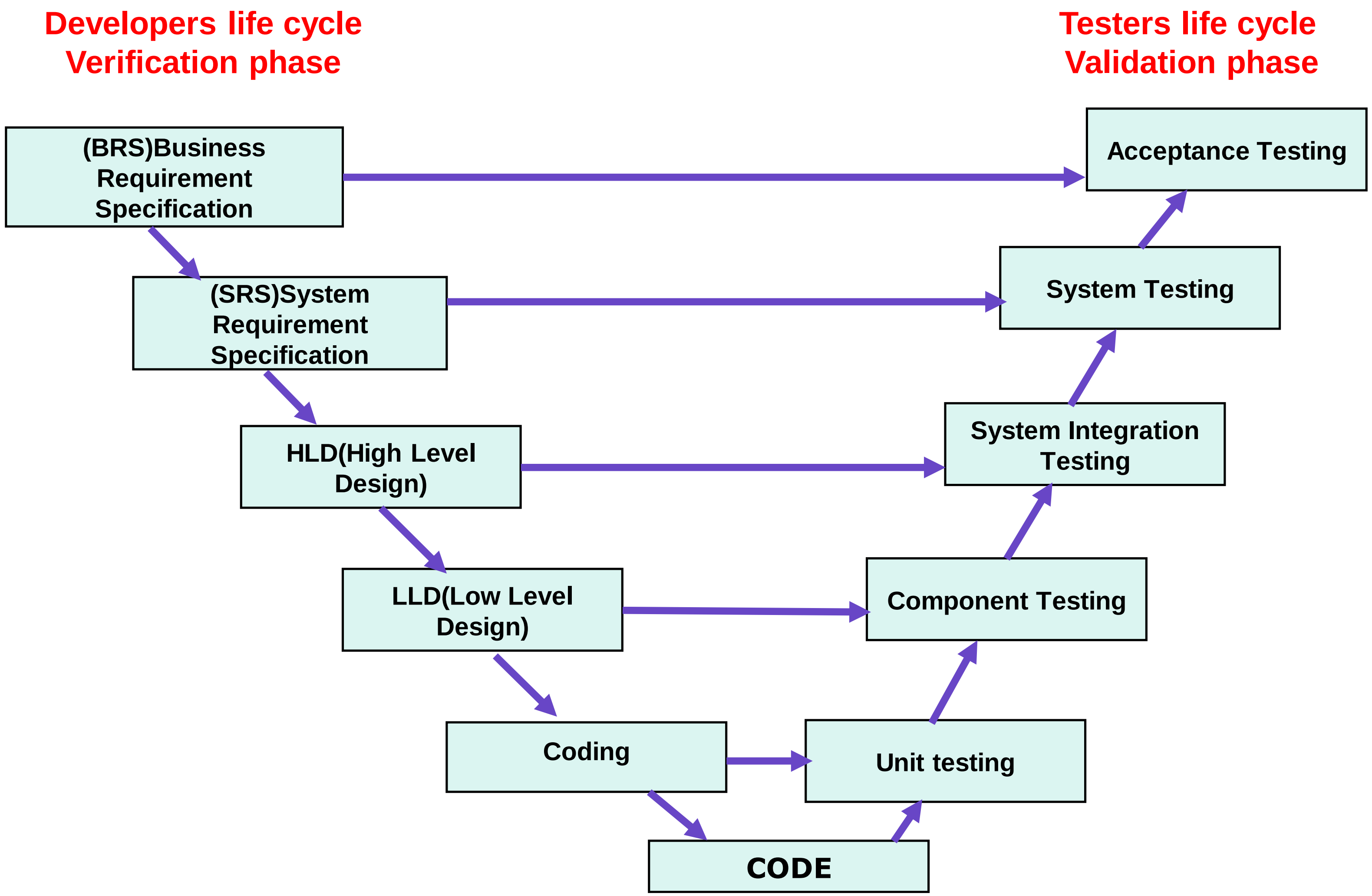
Types of SDLC Models

V model - Verification and Validation model

- **V model** is a software development lifecycle model (SDLC) in which each step executes in a **sequential manner with parallel testing for each development stage.**
- The model is known as the V model because the diagram of the V model is quite similar to the V shape.
- V model is an extension of the Waterfall model, with the improvement that **each development phase will go through a testing phase before moving forward.**
- V model is a **strict model** as the development **only moves forward when the previous step is completed,** and this is made sure by doing testing after each development activity.

Types of SDLC Models

V Model



Types of SDLC Models

V Model : Advantages and Disadvantages

Advantages

- Emphasize planning for verification and validation of the product in early stages of product development
- Each deliverable must be testable
- Project management can track progress by milestones
- Easy to use

Disadvantages

- Does not easily handle concurrent events
- Does not handle iterations or phases
- Does not easily handle dynamic changes in requirements
- Does not contain risk analysis activities

When to use V model?

- Excellent choice for **systems requiring high reliability – Eg. hospital patient control applications, Nuclear Power Stations, Weather Monitoring Etc.,**
- All **requirements are known up-front**
- When it can be modified to **handle changing requirements beyond analysis phase**
- **Solution and technology are known**

Types of SDLC Models

Spiral Model

- The Spiral Model is a risk-driven software development process model that combines elements of both **iterative development and the Waterfall model**.
- It is particularly effective for **large, complex, and high-risk projects**.
- This model organizes the development process into a spiral with **four major phases** repeated in each loop
- Each loop of the spiral represents a phase of the software process and includes activities like **prototyping, risk handling, and customer evaluation**.

Types of SDLC Models

Spiral Model

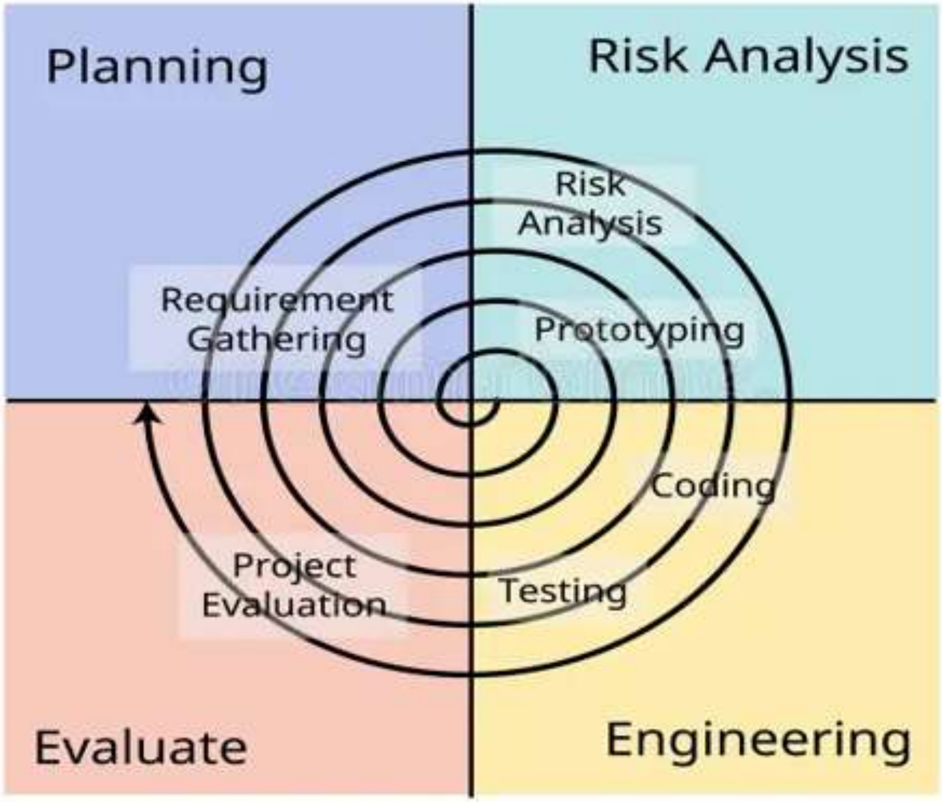
Four major phases repeated in each loop:

1. Planning – Define objectives, identify resources, and gather requirements.

2. Risk Analysis – Evaluate alternatives, identify and mitigate risks.

3. Engineering – Perform design, coding, and testing.

4. Evaluation – Assess the project and decide the next iteration.



Types of SDLC Models

Spiral Model : Advantages and Disadvantages

Advantages

- ❖ Early identification and handling of risks
- ❖ Continuous customer involvement improves accuracy
- ❖ Easily accommodates changes during development
- ❖ Prototyping helps clarify requirements early
- ❖ Well-suited for large and high-risk projects

Disadvantages

- ❖ Not suitable for small or low-budget projects
- ❖ Complex to manage and plan effectively
- ❖ Can be time-consuming with repeated iterations
- ❖ Risk of uncontrolled scope expansion
- ❖ Requires experienced developers and risk analysts

When to use Spiral model?

- Ideal for large, complex, and high-risk projects where **risk management is critical.**
- When the requirements are **unclear or expected to change** frequently during development.
- Suitable when the client needs to **see prototypes** for better understanding and feedback.
- Works best when the development team has experience in **risk analysis** and managing iterative processes.

Types of SDLC Models

Agile Model

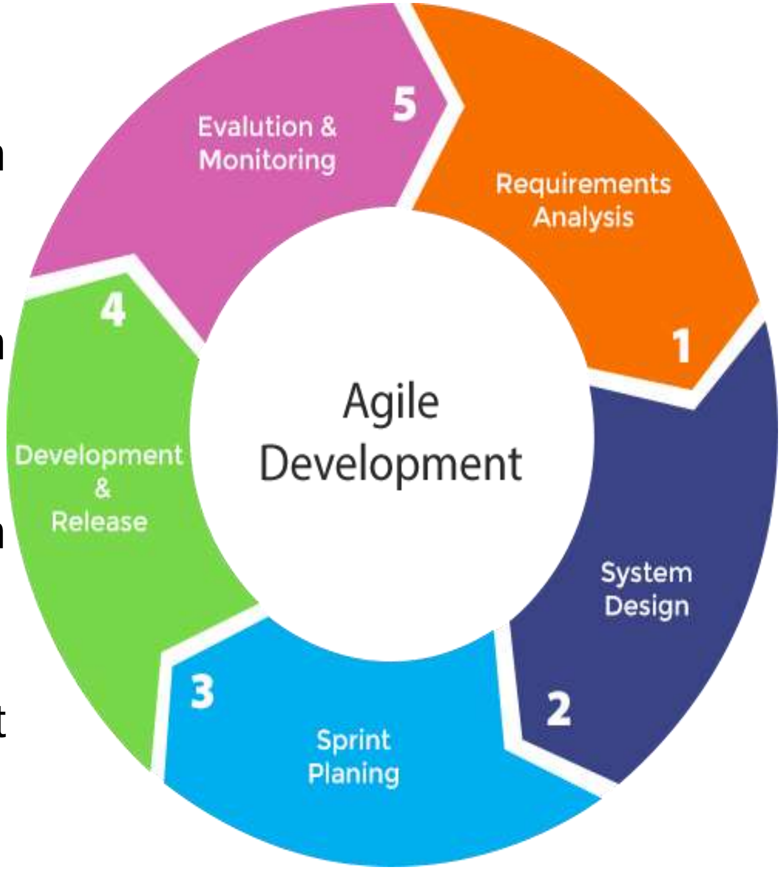
- The Agile Model is a **modern approach to software development** that emphasizes flexibility, collaboration, and customer feedback.
- Unlike traditional linear models, Agile is **iterative and incremental,** allowing teams to deliver working software in **small, manageable parts called sprints.**
- Agile promotes **adaptive planning,** continuous **improvement**, and **quick responses** to change, making it ideal for **dynamic** and **fast-paced projects.**
- It encourages **close interaction** between **developers, testers, and customers** throughout the development process.

Types of SDLC Models

Agile Model

Agile Development Phases

- **Requirements Analysis** – Gather and define user needs through stakeholder collaboration.
- **System Design** – Plan the system architecture and break down features into components.
- **Sprint Planning** – Select tasks for the sprint and assign responsibilities within the team.
- **Development & Release** – Build, test, and deliver a working product increment.
- **Evaluation & Monitoring** – Review progress, collect feedback, and improve in the next sprint.



Agile Model

- The **entire project** is divided into **smaller portions or sprints** with Agile **to reduce project delivery time and hazards.**
- An iteration requires a team to go through the entire software development cycle.
- Within a **single iteration**, an **Agile team** will

  - map out the requirements

  - develop the user stories

  - test their software

  - produce an end deliverable

  - request for user feedback.

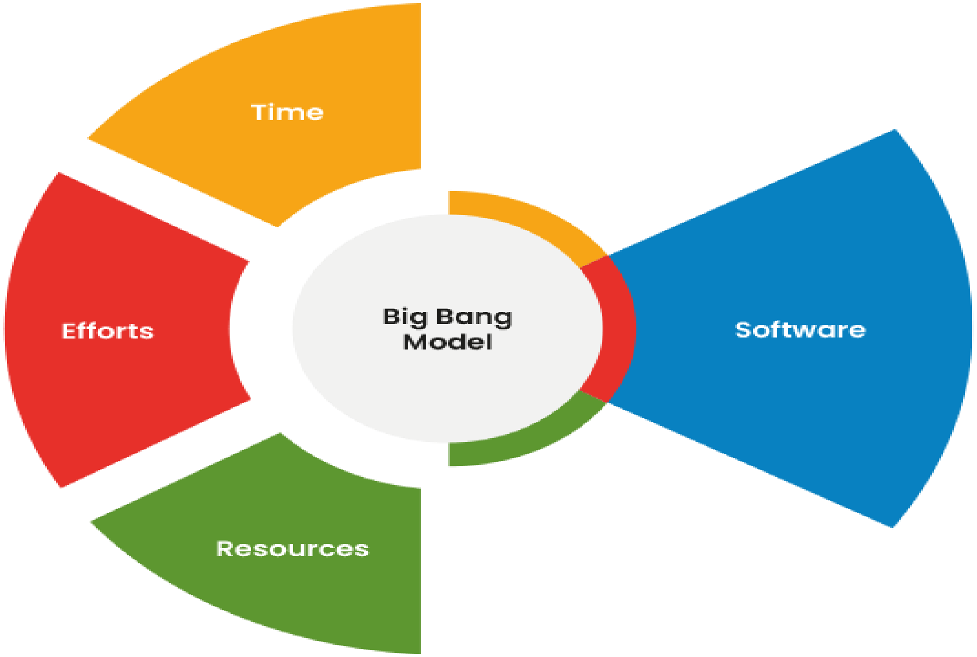
When to use agile model?

- **Requirements** are evolving – when client needs or goals may change frequently.
- **Frequent updates** are needed – when continuous delivery and feedback are essential.
- **Close collaboration** is possible – when customers and developers can interact regularly.
- **Rapid and flexible development** – when early and incremental delivery is important.

Types of SDLC Models

Big Bang Model

- The **Big Bang Model** is a **simple, informal software development model** where development begins **without much planning and everything is developed with minimal process** until the final product is completed — in one "big bang.“
- All **programming effort is focused on coding**, with **little or no formal design or testing during the early stages**.
- The product is developed in **one shot, and the system is integrated and tested only at the end**.



Types of SDLC Models

Big Bang Model : Advantages and Disadvantages

Advantages

- Simple and easy to implement
- No need for initial planning
- Good for learning or experimentation
- Flexible for evolving ideas
- Less documentation overhead

Disadvantages

- High risk of failure.
- Difficult to trace progress or control quality
- Late bug detection leads to costly fixes
- Integration is one big bang at the end
- Final product may not meet user needs

Comparison of SDLC Models

Feature / Model	Waterfall	Iterative	V-Model	Spiral	Agile	Big Bang
Process Flow	Linear, sequential	Cyclic, repeated phases	Linear + parallel testing	Cyclic with risk analysis	Iterative & incremental	No specific process followed
Flexibility	Low	Some changes allowed	Very low	High	Very flexible	Highly flexible
Risk Handling	Poor	Moderate	Poor	Excellent	Good	Very poor
Customer Involvement	Only at start & end	Moderate	Only at start & end	High	Very high	Very low
Cost of Changes	High	Moderate	High	Low to moderate	Low	Very high
Best Suited For	Simple, fixed projects	Evolving projects	Critical systems	High-risk, large projects	Dynamic, fast-changing needs	Small projects or experimental work

Quiz

1) Waterfall model is not suitable

a) Small projects

b)Accommodating Changes

c) Large Projects

d) Maintenance Projects

Answer : Option b)

Quiz

2) Selection of a model is based on

a) Requirements

b)Development team & Users

c) Project type and associated risk

d) All of the mentioned

Answer : Option d)

Quiz

3) What is a software ?

a) Software is set of programs

b) Software is documentation and configuration of data

c) Software is set of programs, documentation & configuration of data

d) Set of Functions

Answer : Option c)

Quiz

4) Actual programming of software code is done during the ____________ step in the SDLC.

a) Maintenance and Evaluation

b) Design

c) Analysis

d) Development and Documentation

Answer : Option d)

Quiz

5) The linear sequential model of software development is

a) A reasonable approach when requirements are well defined

b) A good approach when a working program is required quickly

c) The best approach to use for projects with large development teams

d) An old fashioned model that cannot be used in a modern context

Answer : Option a)

Quiz

6) The largest percentage of total life cycle cost of software is:

a) Design cost

b) Testing cost

c) Coding cost

d) Maintenance cost

Answer : Option d)

Software Testing Levels

Software Testing Levels

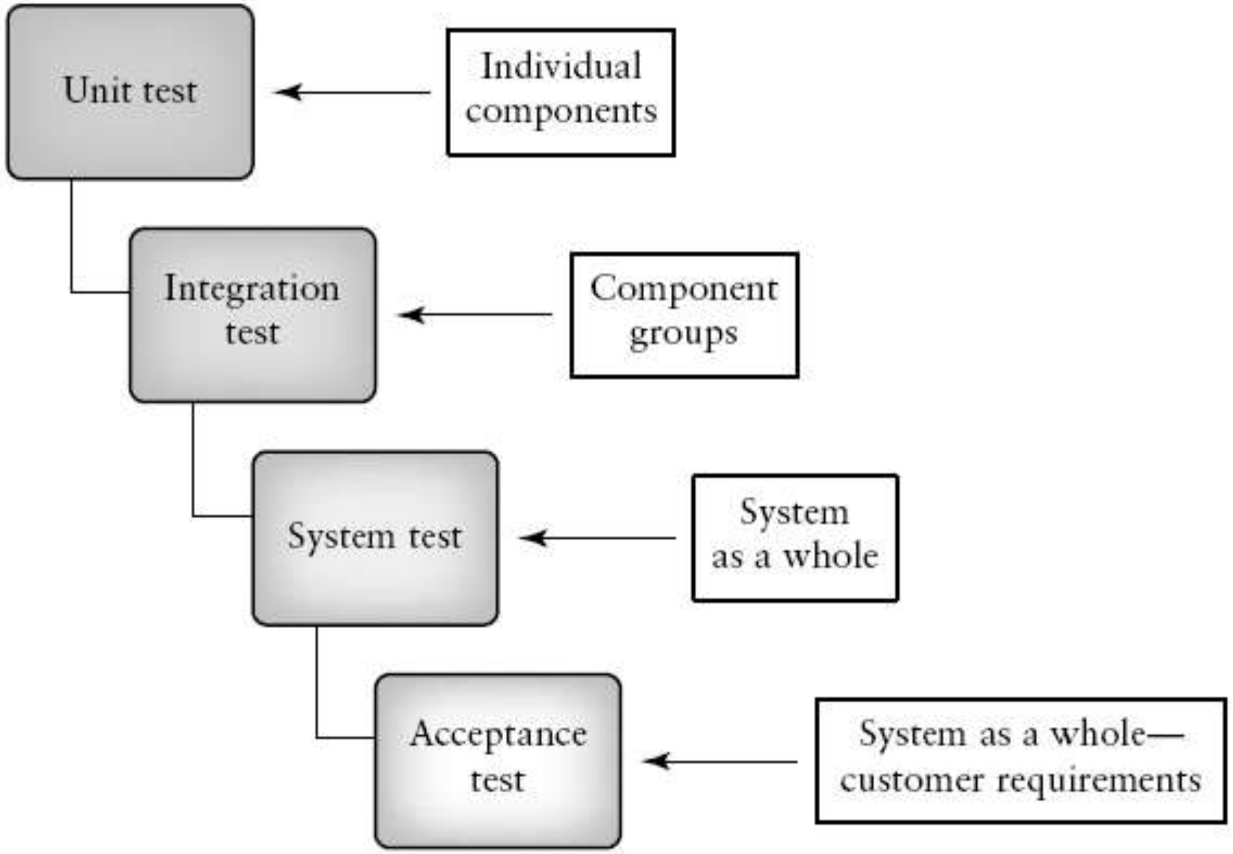
Test Levels

1. Component testing

2. Integration testing

3. System testing

4. Acceptance testing



Component Testing

- Component testing searches for defects in, and verifies the functioning of, software (e.g. modules, programs, objects, classes, etc.) that are separately testable. Stubs, drivers and simulators may be used.
- Component testing occurs with access to the code being tested and with the support of the development environment, such as a unit test framework or debugging tool, and, in practice, usually involves the programmer who wrote the code
- One approach to component testing is to prepare and automate test cases before coding. This is called a test-first approach or test-driven development

Component Testing

Objectives of component testing include:

- Reducing risk
- Verifying whether the functional and non-functional behaviors of the component are as designed and specified
- Building confidence in the component’s quality
- Finding defects in the component
- Preventing defects from escaping to higher test levels

Component Testing

Test basis

Examples of work products that can be used as a test basis for component testing include:

- Detailed design
- Code
- Data model
- Component specifications

Test objects

Typical test objects for component testing include:

- Components, units or modules
- Code and data structures
- Classes
- Database modules

Software Testing Levels

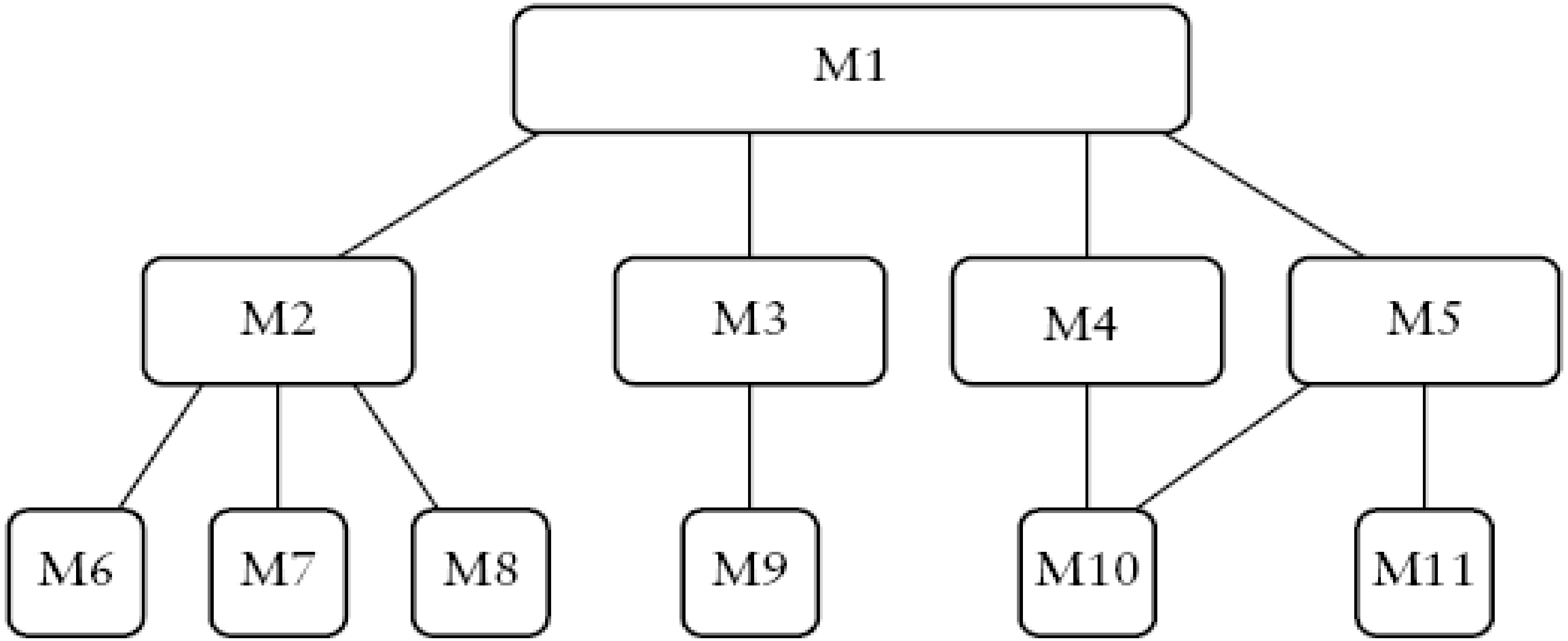
Component Testing

Typical defects and failures

Examples of typical defects and failures for component testing include:

- Incorrect functionality (e.g., not as described in design specifications)
- Data flow problems
- Incorrect code and logic

Integration Testing



Software Testing Levels

Integration Testing

- Integration testing tests interfaces between components, interactions with different parts of a system, such as the operating system, file system, hardware, or interfaces between systems.

Objectives of integration testing include:

- Reducing risk
- Verifying whether the functional and non-functional behaviors of the interfaces are as designed and specified
- Building confidence in the quality of the interfaces
- Finding defects (which may be in the interfaces themselves or within the components or systems)
- Preventing defects from escaping to higher test levels

Integration Testing

- There may be more than one level of integration testing and it may be carried out on test objects of varying size. For example:
- Component integration testing tests the interactions between software components and is done after component testing;
- System integration testing tests the interactions between different systems and may be done after system testing.

Integration Testing (Contd..)

- Systematic integration strategies may be based on the system architecture (such as top-down and bottom-up)
- In order to reduce the risk of late defect discovery, integration should normally be incremental rather than “big bang”.
- At each stage of integration, testers concentrate solely on the integration itself.
- Ideally, testers should understand the architecture and influence integration planning.

Integration Testing (Contd..)

Test basis

Examples of work products that can be used as a test basis for integration testing include:

- Software and system design
- Sequence diagrams
- Interface and communication protocol specifications
- Use cases
- Architecture at component or system level
- Workflows
- External interface definitions

Integration Testing (Contd..)

Test objects

Typical test objects for integration testing include:

- Subsystems
- Databases
- Infrastructure
- Interfaces
- APIs
- Microservices

Integration Testing (Contd..)

Examples of typical defects and failures for component integration testing include:

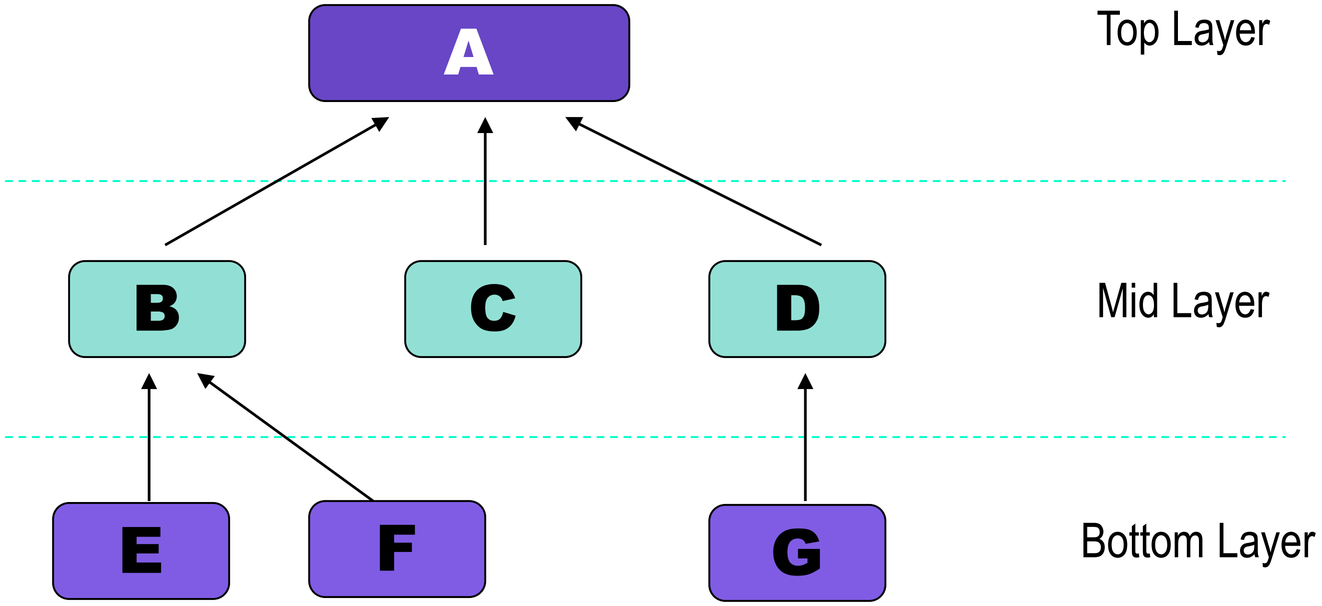
- Incorrect data, missing data, or incorrect data encoding
- Incorrect sequencing or timing of interface calls
- Interface mismatch
- Failures in communication between components
- Unhandled or improperly handled communication failures between components
- Incorrect assumptions about the meaning, units, or boundaries of the data being passed between components

Integration Testing (Contd..)

Examples of typical defects and failures for system integration testing include:

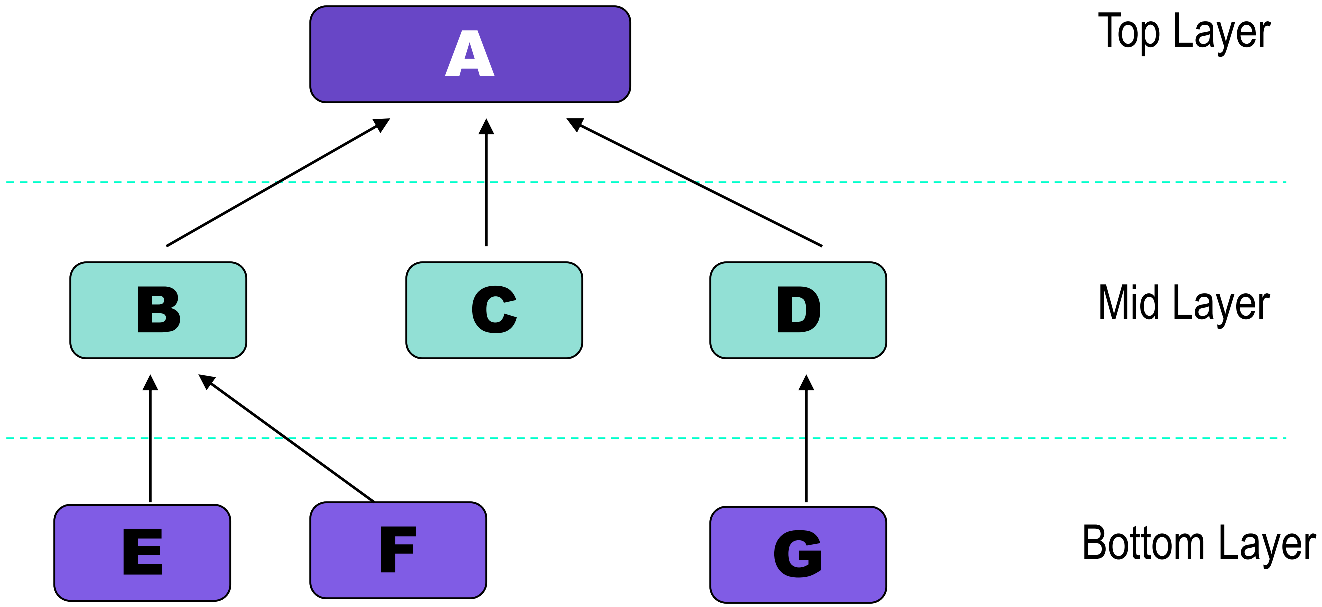
- Inconsistent message structures between systems
- Incorrect data, missing data, or incorrect data encoding
- Interface mismatch
- Failures in communication between systems
- Unhandled or improperly handled communication failures between systems

Top-Down Integration Testing



Test A, Test A+B+C+D, Test All of them together

Bottom-Up Integration Testing



Test B+E+F, Test C, Test D+G, Test All Together

System Testing

- System testing is concerned with the behavior of a whole system/product.
- In system testing, the test environment should correspond to the final target or production environment
- System testing may include tests based on risks and/or on requirements specifications, business processes, use cases
- System testing should investigate both functional and non-functional requirements of the system.
- An independent test team often carries out system testing.

System Testing

Objectives of system testing include:

- Reducing risk
- Verifying whether the functional and non-functional behaviors of the system are as designed and specified
- Validating that the system is complete and will work as expected
- Building confidence in the quality of the system as a whole
- Finding defects
- Preventing defects from escaping to higher test levels or production

System Testing

Test basis

Examples of work products that can be used as a test basis for system testing include:

- System and software requirement specifications (functional and non-functional)
- Risk analysis reports
- Use cases
- Epics and user stories
- Models of system behavior
- State diagrams
- System and user manuals

System Testing

Test objects

Typical test objects for system testing include:

- Applications
- Hardware/software systems
- Operating systems
- System under test (SUT)
- System configuration and configuration data

System Testing

Typical defects and failures

Examples of typical defects and failures for system testing include:

- Incorrect calculations
- Incorrect or unexpected system functional or non-functional behavior
- Incorrect control and/or data flows within the system
- Failure to properly and completely carry out end-to-end functional tasks
- Failure of the system to work properly in the system environment(s)
- Failure of the system to work as described in system and user manuals

Software Testing Levels

Acceptance Testing

- Acceptance testing is often the responsibility of the customers or users of a system; other stakeholders may be involved as well.
- The goal in acceptance testing is to establish confidence in the system, parts of the system or specific non-functional characteristics of the system.

Objectives of acceptance testing include:

- Establishing confidence in the quality of the system as a whole
- Validating that the system is complete and will work as expected
- Verifying that functional and non-functional behaviors of the system are as specified

Acceptance Testing

Common forms of acceptance testing include the following:

- User acceptance testing
- Operational acceptance testing
- Contractual and regulatory acceptance testing
- Alpha and beta testing.

Each is described in the following four subsections.

User acceptance testing (UAT)

- User acceptance testing of the system is typically focused on validating the fitness for use of the system by intended users in a real or simulated operational environment.
- The main objective is building confidence that the users can use the system to meet their needs, fulfill requirements, and perform business processes with minimum difficulty, cost, and risk.

Acceptance Testing

Operational acceptance testing (OAT)

- The acceptance testing of the system by operations or systems administration staff is usually performed in a (simulated) production environment.
- The tests focus on operational aspects, and may include:
  - Testing of backup and restore
  - Installing, uninstalling and upgrading
  - Disaster recovery
  - User management
  - Maintenance tasks

Acceptance Testing

Contractual and regulatory acceptance testing

- Contractual acceptance testing is performed against a contract's acceptance criteria for producing custom-developed software.
- Acceptance criteria should be defined when the parties agree to the contract.
- Contractual acceptance testing is often performed by users or by independent testers.
- Regulatory acceptance testing is performed against any regulations that must be adhered to, such as government, legal, or safety regulations.
- Regulatory acceptance testing is often performed by users or by independent testers, sometimes with the results being witnessed or audited by regulatory agencies.
- The main objective of contractual and regulatory acceptance testing is building confidence that contractual or regulatory compliance has been achieved.

Acceptance Testing

Alpha and beta testing

- Alpha and beta testing are typically used by developers of commercial off-the-shelf (COTS) software who want to get feedback from potential or existing users, customers, and/or operators before the software product is put on the market.
- Alpha testing is performed at the developing organization’s site, not by the development team, but by potential or existing customers, and/or operators or an independent test team.

Acceptance Testing

Alpha and beta testing

- Beta testing is performed by potential or existing customers, and/or operators at their own locations. Beta testing may come after alpha testing, or may occur without any preceding alpha testing having occurred.
- One objective of alpha and beta testing is building confidence among potential or existing customers, and/or operators that they can use the system under normal, everyday conditions, and in the operational environment(s) to achieve their objectives with minimum difficulty, cost, and risk.
- Another objective may be the detection of defects related to the conditions and environment(s) in which the system will be used, especially when those conditions and environment(s) are difficult to replicate by the development team.

Acceptance Testing

Test basis

Examples of work products that can be used as a test basis for any form of acceptance testing include:

- Business processes
- User or business requirements
- Regulations, legal contracts and standards
- Use cases and/or user stories
- System requirements
- System or user documentation
- Installation procedures
- Risk analysis report

Acceptance Testing

Typical test objects

Typical test objects for any form of acceptance testing include:

- System under test
- System configuration and configuration data
- Business processes for a fully integrated system
- Recovery systems and hot sites (for business continuity and disaster recovery testing)
- Operational and maintenance processes
- Forms
- Reports
- Existing and converted production data

Acceptance Testing

Typical defects and failures

Examples of typical defects for any form of acceptance testing include:

- System workflows do not meet business or user requirements
- Business rules are not implemented correctly
- System does not satisfy contractual or regulatory requirements
- Non-functional failures such as security vulnerabilities, inadequate performance efficiency under high loads, or improper operation on a supported platform

Quiz

1) Which of the following shows the correct sequence of testing levels?

a) System → Unit → Integration → Acceptance

b) Acceptance → Unit → Integration → System

c) Integration → Unit → System → Acceptance

d) Unit → Integration → System → Acceptance

Answer : Option d)

Quiz

2) What is the focus of Unit Testing?

a) Business logic as a whole

b) Individual components or functions

c) Entire application

d) External APIs

Answer : Option b)

Quiz

3) What is the main goal of Integration Testing?

a) To verify the login functionality

b) To test if individual units work correctly

c) To ensure that integrated modules communicate correctly

d) To validate user requirements

Answer : Option c)

Quiz

4) System Testing is concerned with:

a) Testing only one module

b) Testing interactions between two modules

c) Testing the entire system as a whole

d) Testing the source code

Answer : Option c)

Quiz

5) Which type of testing is typically performed during System Testing?

a) Regression Testing

b) Performance Testing

c) Functional Testing

d) All of the above

Answer : Option d)

Quiz

6) Acceptance Testing is primarily done to:

a) Fix bugs

b) Verify that code compiles

c) Validate if the system meets business need

d) Test source code

Answer : Option c)

Quiz

7) A bug is found when two modules exchange data. This is most likely a failure in:

a) Integration Testing

b) Acceptance Testing

c) Unit Testing

d) System Testing

Answer : Option a)

Testing Types

Software Testing Levels

Test Types

- Functional Testing
- Non-Functional Testing
- Black Box Testing
- White Box Testing
- Confirmation Testing or Retesting
- Smoke Testing
- Regression Testing
- Usability Testing
- Performance Testing
- Maintenance Testing

Testing of function (Functional Testing)

- Functional tests are based on functions and features and their interoperability with specific systems.
- Functional testing helps in verifying what the system is supposed to do. It aids in testing the product's features or functionality.
- Functional testing requires in-depth customer and product knowledge as well as domain knowledge so as to develop different test cases and find critical defects, as the focus of the testing is to find defects.
- Functional tests may be performed at all test levels.
- Specification-based techniques may be used to derive test conditions and test cases from the functionality of the software or system.
- Functional testing considers the external behavior of the software (black-box testing).

Non-Functional Testing

- It is the testing of “how” the system works”.
- Non-functional testing is performed to verify the quality factors (such as reliability, scalability, Performance efficiency, Compatibility, Usability, Security, Maintainability, Portability etc.,
- These quality factors are also called non-functional requirements. Non-functional testing requires the expected results to be documented in qualitative and quantifiable terms.
- Non-functional testing requires large amount of resources and the results are different for different configurations and resources.

Non-Functional Testing

- ❑Performance testing
- ❑load testing
- ❑stress testing
- ❑usability testing
- ❑maintainability testing
- ❑reliability testing
- ❑portability testing

- Non-functional testing may be performed at all test levels

Functional Versus Non-Functional Testing

Testing aspects	Functional testing	Non-functional testing
Involves	Product features and functionality	Quality factors
Tests	Product behavior	Behavior and experience
Result conclusion	Simple steps written to check expected results	Huge data collected and analyzed
Results varies due to	Product implementation	Product implementation, resources, and configurations
Testing focus	Defect detection	Qualification of product
Knowledge required	Product and domain	Product, domain, design, architecture, statistical skills
Failures normally due to	Code	Architecture, design, and code
Test case repeatability	Repeated many times	Repeated only in case of failures and for different configurations
Configuration	One-time setup for a set of test cases	Configuration changes for each test case

White Box Testing (Structural Testing)

- White Box Testing tests the internal structure, logic, and code of the software. The tester knows the internal code and uses that knowledge to design test cases.

How It Works:

- Requires programming knowledge
- Focuses on paths, branches, loops, and conditions inside the code
- Test cases are written to cover all logical paths

White Box Testing (Structural Testing)

Techniques:

- Control Flow Testing
- Statement Coverage
- Branch/Decision Coverage
- Path Coverage
- Loop Testing

Example: A function calculates a discount. The tester writes test cases for all if-else conditions in the logic to ensure coverage.

When to Use: Unit testing, Testing algorithm-heavy components, Performance optimization and code security

Black Box Testing (Behavioral Testing)

- Black Box Testing tests the functionality of an application without knowing its internal code or structure.

How It Works:

- Testers focus on inputs and expected outputs
- Test cases are based on requirements and specifications
- Does not require coding knowledge

Black Box Testing (Behavioral Testing)

Techniques:

- Equivalence Partitioning
- Boundary Value Analysis
- Decision Table Testing
- State Transition Testing
- Use Case Testing

Example: Login screen: Tester checks that the system allows login with valid credentials and shows an error with invalid ones.

When to Use: System testing, User Acceptance Testing (UAT), Functional and non-functional testing

Grey Box Testing

- Grey Box Testing is a hybrid approach, where the tester has partial knowledge of the internal structure but primarily tests based on the external behavior.

How It Works:

- Combines benefits of White Box and Black Box testing
- Testers may review code or architecture before testing
- Aimed at finding issues missed by either approach alone

Grey Box Testing

Example:

Testing a shopping cart: The tester knows the database schema and checks if item quantities update correctly, while testing through the UI.

When to Use:

- Integration testing
- Web services testing (e.g., API and frontend)
- Security and penetration testing

Confirmation Testing

- Changes are typically made to a component or system to either enhance it by adding a new feature or to fix it by removing a defect.
- Testing should then also include confirmation testing and regression testing.
- Confirmation testing confirms that an original defect has been successfully fixed. Depending on the risk, one can test the fixed version of the software in several ways, including:
- executing all tests that previously have failed due to the defect, or, also by
- adding new tests to cover any changes that were needed to fix the defect

Confirmation Testing

- However, when time or money is short when fixing defects, confirmation testing might be restricted to simply exercising the test steps that should reproduce the failure caused by the defect and checking that the failure does not occur.

Regression Testing

- Regression testing is testing of unchanged features of the application to make sure that any bug fixes, adding new features, deleting, or updating existing features, are not impacting the working application.
- To find out regression scope is an important part in Regression Testing.
- To find out regression scope, Tester needs to find out the area of application where changes happened and the Impact of those changes on the entire application.
- It is difficult to cover the whole regression test suite in every release, so Automation Testing Tools are used in regression testing.

Regression Testing Versus Confirmation

Regression Testing	Confirmation/ Retesting
Performed after code-changes to ensure previously working features work fine.	Performed to check if the bug-fix is valid or not.
It involves executing either all the test cases or the limited set of test cases that cover all the features (regression tests).	It involves executing only the failed tests that are fixed by the developers
We can say that it has a lower priority than retesting. Since, we are just making sure that the test cases that passed previously, still get passed.	Retesting can be considered as of higher priority than regression because we need to make sure that the previously failed tests got fixed or not after the bug fixes.

Smoke Testing

- Smoke testing is performed to verify that basic and critical functionality of the system under test is working fine at a very high level.
- Whenever a new build is provided by the development team, then the Software Testing team validates the build and ensures that no major issue exists. The testing team will ensure that the build is stable, and a detailed level of testing will be carried out further.
- For example, tester is testing pet insurance website. Buying an insurance policy, adding another pet, providing quotes are all basic and critical functionality of the application. Smoke testing for this website verifies that all these functionalities are working fine before doing any in-depth testing.

Smoke Testing

Smoke testing consists of :

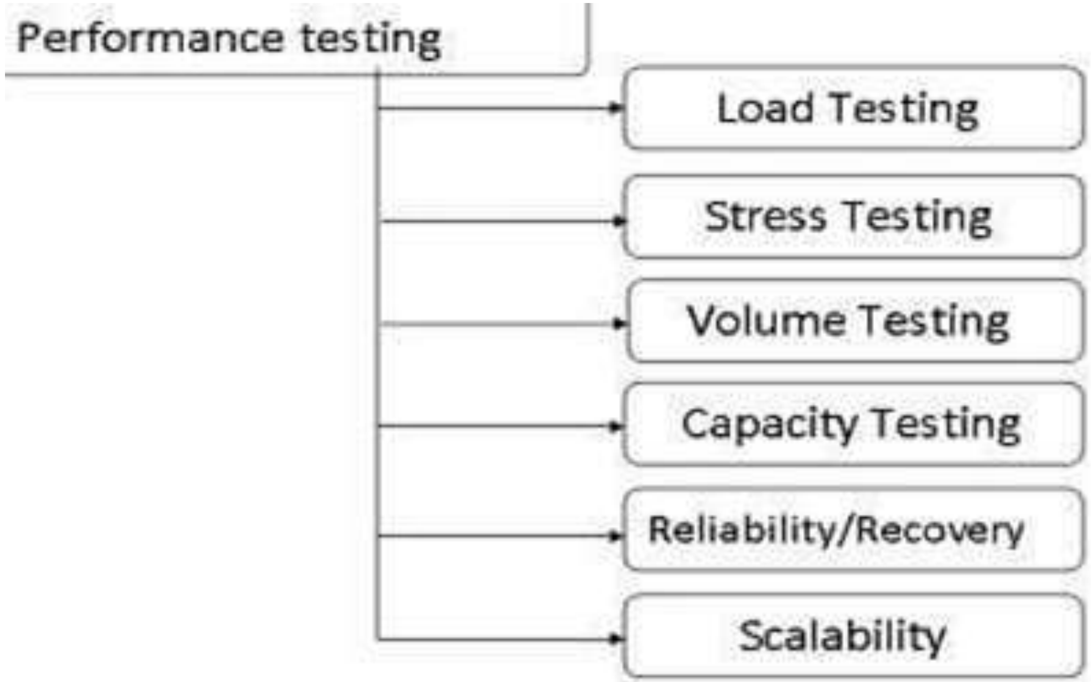
1. Identifying the basic functionality that a product must satisfy;
2. Designing test cases to ensure that these basic functionality work and packaging them into a smoke test suite;
3. Ensuring that every time a product is built, this suite is run successfully before anything else is run; and
4. If this suite fails, escalating to the developers to identify the changes and perhaps change or roll back the changes to a state where the smoke test suite succeeds.

Usability Testing

- It is a non-functional type of software testing. It is broadly divided into understandability, learnability, operability, attractiveness, and compliance. Usability testing is to determine the extent to which we understand the software product, easy to learn, easy to operate, and attractive to the users under specified conditions and requirements.
- This type of testing is usually executed by real-life users and not the development team. The development team is the one who has created the product, and hence they fail to find fewer defects that are related to user experience.
- Usability testing is mainly divided into three categories. These are:
  - –Explorative
  - –Assessment
  - –Comparative

Performance Testing

- Performance Testing also knows as 'Perf Testing', is a type of testing performed to check how application or software performs under workload in terms of responsiveness and stability.
- The Performance Test goal is to identify and remove performance bottlenecks from an application.
- This test is mainly performed to check whether the software meets the expected requirements for application speed, scalability, and stability.



Load Testing

- Load Testing is a type of performance test where the application is tested for its performance on normal and peak usage. The performance of an application is checked with respect to its response to the user request and its ability to respond consistently within an accepted tolerance on different user loads.
- The key considerations are:

1. What is the maximum load the application is able to hold before the application starts behaving unexpectedly?

Load Testing

1. How much data the Database able to handle before the system slows or the crash is observed?

2. Are there any network related issues to be addressed?

- For example, your application handles 100 users at a time with a response time of 3 seconds, then load testing can be done by applying a load of the maximum of 100 or less than 100 users. The goal is to verify that the application is responding within 3 seconds for all the users.

Stress Testing

- Stress Testing is used to find ways to break the system.
- The test also provides the range of maximum load the system can hold.
- Generally, Stress Testing has an incremental approach where the load is increased gradually. The test is started with a load for which the application has already been tested.
- Then, more load is added slowly to stress the system. The point at which we start seeing servers not responding to the requests is considered the breaking point.

Stress Testing

The following questions are to be addressed:

- What is the maximum load a system can sustain before it breaks down?
- How is the system break down?
- Is the system able to recover once it’s crashed?
- In how many ways a system can break and which are the weak node while handling the unexpected load?
- For example, your application handles 1000 users at a time with a response time of 4 seconds, then stress testing can be done by applying a load of more than 1000 users. Test the application with 1100,1200,1300 users and notice the response time. The goal is to verify the stability of an application under stress.

Volume Testing

- Volume Testing is to verify that the performance of the application is not affected by the volume of data that is being handled by the application. In order to execute a Volume Test, a huge volume of data is entered into the database. This test can be an incremental or steady test. In the incremental test, the volume of data is increased gradually.
- Generally, with the application usage, the database size grows, and it is necessary to test the application against a heavy database. A good example of this could be a website of a new school or a college having small amounts of data to store initially, but after 5-10 years, the data stores in the database of the website is much more.

Capacity Testing

- Is the application capable of meeting business volume under both normal and peak load conditions?
- Capacity Testing is generally done for future prospects. Capacity Testing addresses the following:

1. Will the application be able to support the future load?
2. Is the environment capable of standing for the upcoming increased load?

Capacity Testing

3. What are the additional resources required to make the environment capable enough?

- Capacity Testing is used to determine how many users and/or transactions a given web application will support and still meet performance. During this testing, resources such as processor capacity, network bandwidth, memory usage, disk capacity, etc. are considered and altered to meet the goal.
- Online Banking is a perfect example of where capacity testing could play a major role.

Reliability/Recovery Testing

- Reliability Testing or Recovery Testing – is to verify whether or not the application is able to return back to its normal state after a failure or abnormal behavior and how long does it take for it to do so (in other words, time estimation).
- If an online trading site experiences a failure where the users are not able to buy/sell shares at a certain point of the day (peak hours) but are able to do so after an hour or two, we can say the application is reliable or recovered from the abnormal behavior.

Scalability Testing

- Scalability testing is testing an application’s stability and response time by applying load, which is more than the designed number of users for an application.
- For example, your application handles 1000 users at a time with a response time of 2 seconds, then scalability testing can be done by applying a load of more than 1000 users and gradually increasing the number of users to find out where exactly my application is crashing.
- Let’s say my application is giving response time as follows:
  - 1000 users -2 sec
  - 1400 users -2 sec
  - 4000 users -3 sec
  - 5000 users -45 sec
  - 5150 users- crash – This is the point that needs to identify in scalability testing

Maintenance Testing

- Once deployed to production environments, software and systems need to be maintained.
- Changes of various sorts are almost inevitable in delivered software and systems, either to fix defects discovered in operational use, to add new functionality, or to delete or alter already-delivered functionality.
- Maintenance is also needed to preserve or improve non-functional quality characteristics of the component or system over its lifetime, especially performance efficiency, compatibility, reliability, security, , and portability.

Maintenance Testing

- When any changes are made as part of maintenance, maintenance testing should be performed, both to evaluate the success with which the changes were made and to check for possible side-effects (e.g., regressions) in parts of the system that remain unchanged (which is usually most of the system). Maintenance can involve planned releases and unplanned releases (hot fixes).
- A maintenance release may require maintenance testing at multiple test levels, using various test types, based on its scope.

The scope of maintenance testing depends on:

- The degree of risk of the change, for example, the degree to which the changed area of software communicates with other components or systems.
- The size of the existing system.
- The size of the change

Maintenance Testing

The triggers for maintenance and maintenance testing can be classified as follows:

- Modifications, such as planned enhancements (i.e., release-based), corrective changes or hot fixes.
- Upgrades or migrations of the operational environment, such as from one platform to another, which can require tests associated with the new environment as well as of the changed software, or tests of data conversion when data from another application is migrated into the system being maintained.
- Retirement, such as when an application reaches the end of its life. When a system is retired, this can require testing of data archiving if long data retention periods are required. Testing of restore and retrieval procedures after archiving may also be needed in the event that certain data is required during the archiving period.