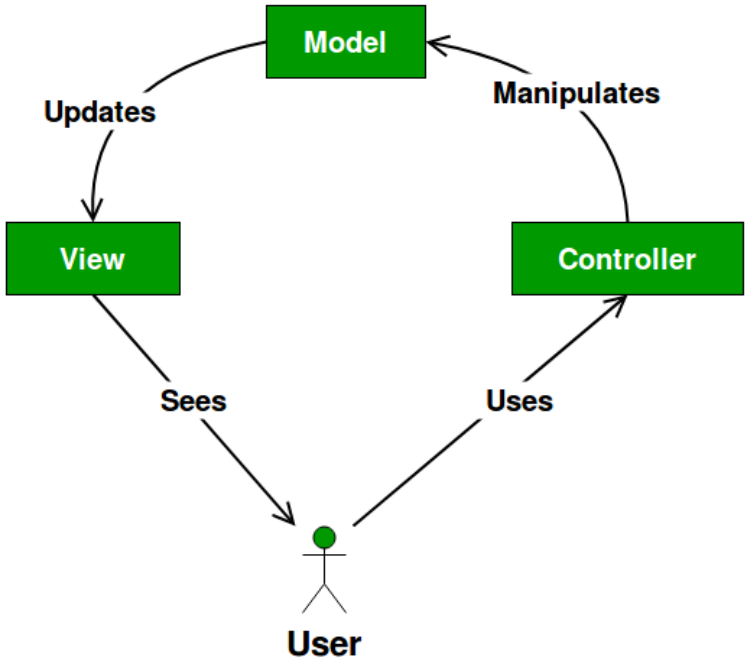
The **Model View Controller** (MVC) design pattern specifies that an application consist of a data model, presentation information, and control information.

UML Diagram MVC Design Pattern



Design components

- The **Model** contains only the pure application data, it contains no logic describing how to present the data to a user.
- The **View** presents the model's data to the user. The view knows how to access the model's data, but it does not know what this data means or what the user can do to manipulate it. View just represent, displays the application's data on screen.
- The **Controller** exists between the view and the model. It is where the actual business logic is written. It listens to events triggered by the view (or another external source) and executes the appropriate reaction to these events.

Advantages

- Multiple developers can work simultaneously on the model, controller and views.
- MVC enables logical grouping of related actions on a controller together. The views for a specific model are also grouped together.
- Models can have multiple views.
- The overall components of an application are easily manageable & are less dependent on each other for proper functioning of application.

Disadvantages

- The framework navigation can be complex because it introduces new layers of abstraction and requires users to adapt to the decomposition criteria of MVC.

- Knowledge on multiple technologies becomes the norm. Developers using MVC need to be skilled in multiple technologies.

- Sequence diagram: an "interaction diagram" that models a single scenario executing in a system.
- **Cost estimation** simply means a technique that is used to find out the cost estimates

COCOMO Model

Cocomo (Constructive Cost Model) is a regression model based on LOC, i.e **number of Lines of Code**. It is a procedural cost estimate model for software projects and is often used as a process of reliably predicting the various parameters associated with making a project such as size, effort, cost, time, and quality. It was proposed by Barry Boehm in 1981 and is based on the study of 63 projects, which makes it one of the best-documented models.

The key parameters which define the quality of any software products, which are also an outcome of the Cocomo are primarily Effort & Schedule:

- **Effort:** Amount of labor that will be required to complete a task. It is measured in person-months units.
- **Schedule:** Simply means the amount of time required for the completion of the job, which is, of course, proportional to the effort put in. It is measured in the units of time such as weeks, and months.

These characteristics

1. Organic – A software project is said to be an organic type if the team size required is adequately small, the problem is well understood and has been solved in the past and also the team members have a nominal experience regarding the problem.

2. Semi-detached – A software project is said to be a Semi-detached type if the vital characteristics such as team size, experience, and knowledge of the various programming environment lie in between that of organic and Embedded.

3. Embedded – A software project requiring the highest level of complexity, creativity, and experience requirement fall under this category. Such software requires a larger team size than the other two models and also the developers need to be sufficiently experienced and creative to develop such complex models.

1. Basic COCOMO Model
2. Intermediate COCOMO Model
3. Detailed COCOMO Model

Advantages of the COCOMO model:

1. Provides a systematic way to estimate the cost and effort of a software project.
2. Can be used to estimate the cost and effort of a software project at different stages of the development process.
3. Helps in identifying the factors that have the greatest impact on the cost and effort of a software project.

4. Can be used to evaluate the feasibility of a software project by estimating the cost and effort required to complete it.

Disadvantages of the COCOMO model:

1. Assumes that the size of the software is the main factor that determines the cost and effort of a software project, which may not always be the case.
2. Does not take into account the specific characteristics of the development team, which can have a significant impact on the cost and effort of a software project.
3. Does not provide a precise estimate of the cost and effort of a software project, as it is based on assumptions and averages.

- COCOMO 2 incorporates a range of sub-models that produce increasingly detailed software estimates.
- The sub-models in COCOMO 2 are:
  - **Application composition model.** Used when software is composed from existing parts.
  - **Early design model**. Used when requirements are available but design has not yet started.
  - **Reuse model.** Used to compute the effort of integrating reusable components.
  - **Post-architecture model**. Used once the system architecture has been designed and more information about the system is available.

Lines of Code (LOC) in Software Engineering

A **line of code (LOC)** is any line of text in a code that is not a comment or blank line, and also header lines, in any case of the number of statements or fragments of statements on the line. LOC clearly consists of all lines containing the declaration of any variable, and executable and non-executable statements. As Lines of Code (LOC) only counts the volume of code, you can only use it to compare or estimate projects that use the same language and are coded using the same coding standards.

Features :

- Variations such as “source lines of code”, are used to set out a codebase.
- LOC is frequently used in some kinds of arguments.
- They are used in assessing a project’s performance or efficiency.

Advantages :

- Most used metric in cost estimation.
- Its alternates have many problems as compared to this metric.
- It is very easy in estimating the efforts.

Disadvantages :

- Very difficult to estimate the LOC of the final program from the problem specification.
- It correlates poorly with quality and efficiency of code.
- It doesn’t consider complexity.

Functional Point (FP) Analysis

Allan J. Albrecht initially developed function Point Analysis in 1979 at IBM and it has been further modified by the International Function Point Users Group (IFPUG). FPA is used to make estimate of the software project, including its testing in terms of functionality or function size of the software product. However, functional point analysis may be used for the test estimation of the product. The functional size of the product is measured in terms of the function point, which is a standard of measurement to measure the software application.

Objectives of FPA

The basic and primary purpose of the functional point analysis is to measure and provide the software application functional size to the client, customer, and the stakeholder on their request. Further, it is used to measure the software project development along with its maintenance, consistently throughout the project irrespective of the tools and the technologies.

Differentiate between FP and LOC

FP	LOC
1. FP is specification based.	1. LOC is an analogy based.
2. FP is language independent.	2. LOC is language dependent.
3. FP is user-oriented.	3. LOC is design-oriented.
4. It is extendible to LOC.	4. It is convertible to FP (backfiring)

Pricing to win

- The project costs whatever the customer has to spend on it.
- Advantages:
  - You get the contract.
- Disadvantages:

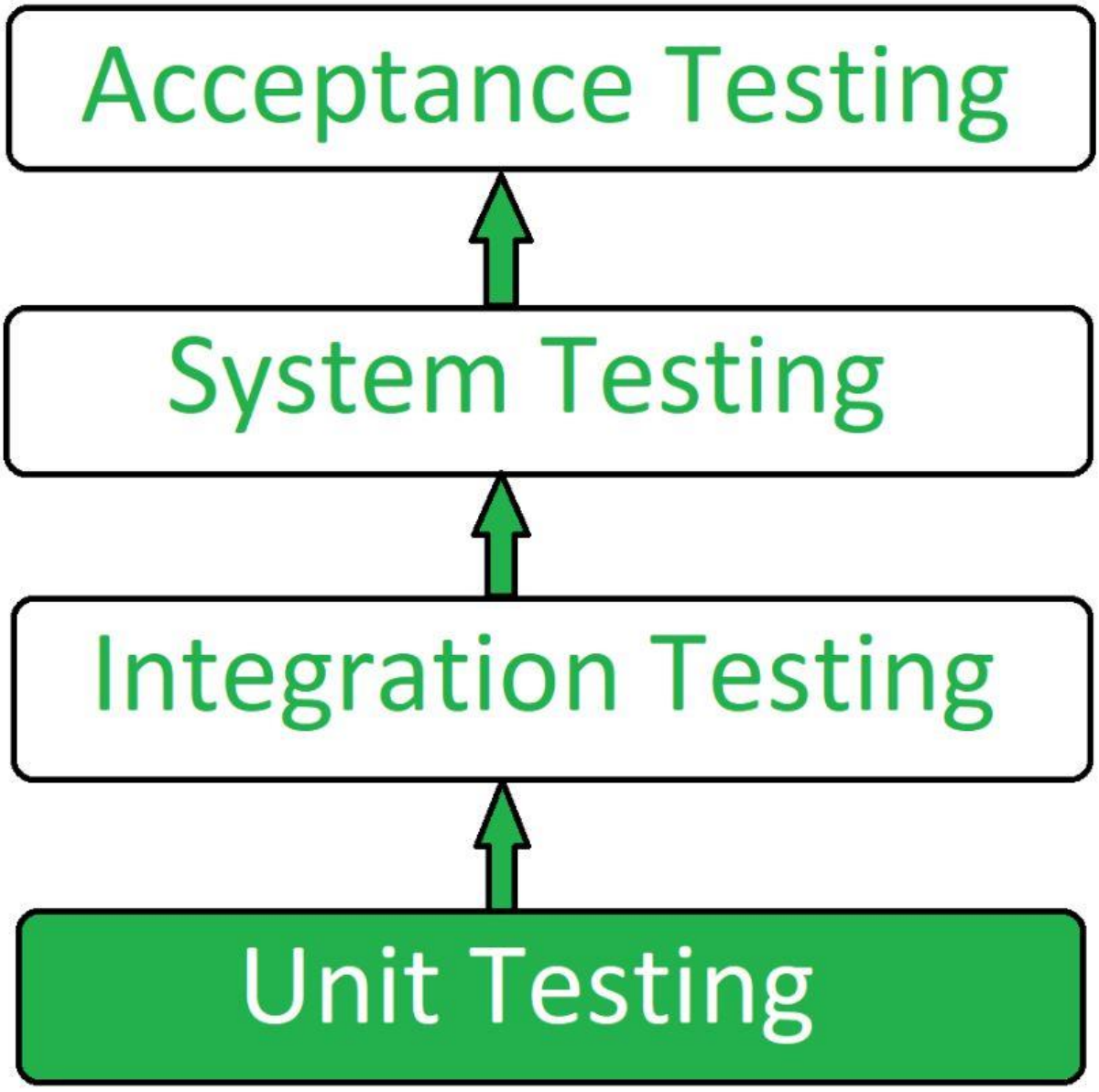
- The probability that the customer gets the system he or she wants is small. Costs do not accurately reflect the work required.

Unit Testing | Software Testing

Unit testing is a type of software testing that focuses on individual units or components of a software system

Objective of Unit Testing:

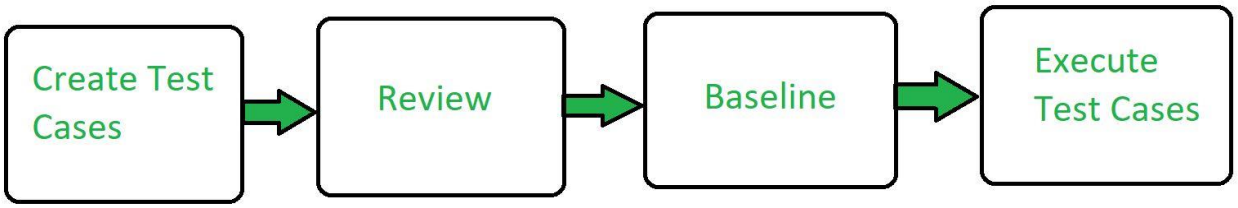
1. To isolate a section of code.
2. To verify the correctness of the code.
3. To test every function and procedure.
4. To fix bugs early in the development cycle and to save costs.
5. To help the developers to understand the code base and enable them to make changes quickly.
6. To help with code reuse.



Types of Unit Testing:

There are 2 types of Unit Testing: **Manual**, and **Automated**.

Workflow of Unit Testing:



Unit Testing Techniques:

There are 3 types of Unit Testing Techniques. They are

1. **Black Box Testing:** This testing technique is used in covering the unit tests for input, user interface, and output parts.
2. **White Box Testing:** This technique is used in testing the functional behavior of the system by giving the input and checking the functionality output including the internal design structure and code of the modules.
3. **Gray Box Testing:** This technique is used in executing the relevant test cases, test methods, test functions, and analyzing the code performance for the modules.

Unit Testing Tools:

Advantages of Unit Testing:

1. Unit Testing allows developers to learn what functionality is provided by a unit and how to use it to gain a basic understanding of the unit API.
2. Unit testing allows the programmer to refine code and make sure the module works properly.
3. Unit testing enables testing parts of the project without waiting for others to be completed.
4. Early Detection of Issues: Unit testing allows developers to detect and fix issues early in the development process, before they become larger and more difficult to fix.
5. Improved Code Quality: Unit testing helps to ensure that each unit of code works as intended and meets the requirements, improving the overall quality of the software.
6. Increased Confidence: Unit testing provides developers with confidence in their code, as they can validate that each unit of the software is functioning as expected.
7. Faster Development: Unit testing enables developers to work faster and more efficiently, as they can validate changes to the code without having to wait for the full system to be tested.

8. Better Documentation: Unit testing provides clear and concise documentation of the code and its behavior, making it easier for other developers to understand and maintain the software.
9. Facilitation of Refactoring: Unit testing enables developers to safely make changes to the code, as they can validate that their changes do not break existing functionality.
10. Reduced Time and Cost: Unit testing can reduce the time and cost required for later testing, as it helps to identify and fix issues early in the development process.

Disadvantages of Unit Testing:

1. The process is time-consuming for writing the unit test cases.
2. Unit Testing will not cover all the errors in the module because there is a chance of having errors in the modules while doing integration testing.
3. Unit Testing is not efficient for checking the errors in the UI(User Interface) part of the module.
4. It requires more time for maintenance when the source code is changed frequently.
5. It cannot cover the non-functional testing parameters such as scalability, the performance of the system, etc.
6. Time and Effort: Unit testing requires a significant investment of time and effort to create and maintain the test cases, especially for complex systems.
7. Dependence on Developers: The success of unit testing depends on the developers, who must write clear, concise, and comprehensive test cases to validate the code.
8. Difficulty in Testing Complex Units: Unit testing can be challenging when dealing with complex units, as it can be difficult to isolate and test individual units in isolation from the rest of the system.
9. Difficulty in Testing Interactions: Unit testing may not be sufficient for testing interactions between units, as it only focuses on individual units.
10. Difficulty in Testing User Interfaces: Unit testing may not be suitable for testing user interfaces, as it typically focuses on the functionality of individual units.
11. Over-reliance on Automation: Over-reliance on automated unit tests can lead to a false sense of security, as automated tests may not uncover all possible issues or bugs.
12. Maintenance Overhead: Unit testing requires ongoing maintenance and updates, as the code and test cases must be kept up-to-date with changes to the software.

Difference between Alpha and Beta Testing

Alpha Testing is a type of software testing performed to identify bugs before releasing the product to real users or to the public. Alpha Testing is one of the user acceptance tests.

Beta Testing is performed by real users of the software application in a real environment. Beta testing is one type of User Acceptance Testing.

Difference between Alpha and Beta Testing:

Alpha Testing	**Beta Testing**
Alpha testing involves both the white box and black box testing.	Beta testing commonly uses black-box testing.
Alpha testing is performed by testers who are usually internal employees of the organization.	Beta testing is performed by clients who are not part of the organization.
Alpha testing is performed at the developer's site.	Beta testing is performed at the end-user of the product.
Reliability and security testing are not checked in alpha testing.	Reliability, security and robustness are checked during beta testing.
Alpha testing ensures the quality of the product before forwarding to beta testing.	Beta testing also concentrates on the quality of the product but collects users input on the product and ensures that the product is ready for real time users.
Alpha testing requires a testing environment or a lab.	Beta testing doesn't require a testing environment or lab.
Alpha testing may require a long execution cycle.	Beta testing requires only a few weeks of execution.

Alpha Testing	Beta Testing
Developers can immediately address the critical issues or fixes in alpha testing.	Most of the issues or feedback collected from the beta testing will be implemented in future versions of the product.
Multiple test cycles are organized in alpha testing.	Only one or two test cycles are there in beta testing.

Differences between Verification and Validation

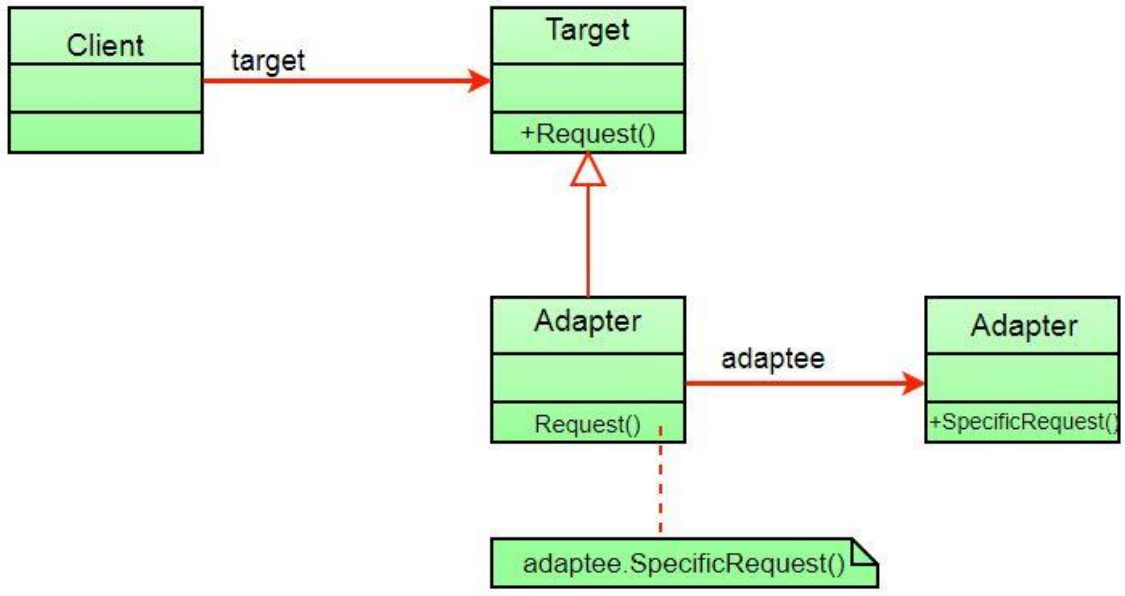
Verification is the process of checking that a software achieves its goal without any bugs. It is the process to ensure whether the product that is developed is right or not. It verifies whether the developed product fulfills the requirements that we have. Verification is static testing.
Verification means **Are we building the product right?**
Validation is the process of checking whether the software product is up to the mark or in other words product has high level requirements. It is the process of checking the validation of product i.e. it checks what we are developing is the right product. it is validation of actual and expected product. Validation is the dynamic testing.
Validation means **Are we building the right product?**
The difference between Verification and Validation is as follow:

Verification	Validation
It includes checking documents, design, codes and programs.	It includes testing and validating the actual product.
Verification is the static testing.	Validation is the dynamic testing.

Verification	Validation
It does *not* include the execution of the code.	It includes the execution of the code.
Methods used in verification are reviews, walkthroughs, inspections and desk-checking.	Methods used in validation are Black Box Testing, White Box Testing and non-functional testing.
It checks whether the software conforms to specifications or not.	It checks whether the software meets the requirements and expectations of a customer or not.
It can find the bugs in the early stage of the development.	It can only find the bugs that could not be found by the verification process.
The goal of verification is application and software architecture and specification.	The goal of validation is an actual product.
Quality assurance team does verification.	Validation is executed on software code with the help of testing team.
It comes before validation.	It comes after verification.
It consists of checking of documents/files and is performed by human.	It consists of execution of program and is performed by computer.
Verification refers to the set of activities that ensure software correctly implements the specific function.	Validation refers to the set of activities that ensure that the software that has been built is traceable to customer requirements.
After a valid and complete specification the verification starts.	Validation begins as soon as project starts.
Verification is for prevention of errors.	Validation is for detection of errors.

Verification	Validation
Verification is also termed as white box testing or static testing as work product goes through reviews.	Validation can be termed as black box testing or dynamic testing as work product is executed.
Verification finds about 50 to 60% of the defects.	Validation finds about 20 to 30% of the defects.
Verification is based on the opinion of reviewer and may change from person to person.	Validation is based on the fact and is often stable.
Verification is about process, standard and guideline.	Validation is about the product.

Definition: The adapter pattern convert the interface of a class into another interface clients expect. Adapter lets classes work together that couldn't otherwise because of incompatible interfaces. **Class Diagram:**



The client sees only the target interface and not the adapter. The adapter implements the target interface. Adapter delegates all requests to Adaptee. **Example:**

Suppose you have a Bird class with fly() , and makeSound()methods. And also a ToyDuck class with squeak() method. Let's assume that you are short on ToyDuck objects and you would like to use Bird objects in their place. Birds have some similar functionality but implement a different interface, so we can't use them directly. So we will use adapter pattern. Here our client would be ToyDuck and adaptee would be Bird.

Advantages:

- Helps achieve reusability and flexibility.
- Client class is not complicated by having to use a different interface and can use polymorphism to swap between different implementations of adapters.

Disadvantages:

- All requests are forwarded, so there is a slight increase in the overhead.
- Sometimes many adaptations are required along an adapter chain to reach the type which is required.

Risk Management

- ✧ Risk management is concerned with identifying risks and drawing up plans to minimise their effect on a project.

A software project can be concerned with a large variety of risks. In order to be adept to systematically identify the significant risks which might affect a software project, it is essential to classify risks into different classes. The project manager can then check which risks from each class are relevant to the project.

There are three main classifications of risks which can affect a software project:

1. Project risks
2. Technical risks
3. Business risks

1. Project risks: Project risks concern differ forms of budgetary, schedule, personnel, resource, and customer-related problems. A vital project risk is schedule slippage. Since the software is intangible, it is very tough to monitor and control a software project. It is very tough to control something which cannot be identified. For any manufacturing program, such as the manufacturing of cars, the plan executive can recognize the product taking shape.

2. Technical risks: Technical risks concern potential method, implementation, interfacing, testing, and maintenance issue. It also consists of an ambiguous specification, incomplete specification, changing specification, technical uncertainty, and technical obsolescence. Most technical risks appear due to the development team's insufficient knowledge about the project.

3. Business risks: This type of risks contain risks of building an excellent product that no one need, losing budgetary or personnel commitments, etc.

Other risk categories

1. **1. Known risks:** Those risks that can be uncovered after careful assessment of the project program, the business and technical environment in which the plan is being developed, and more reliable data sources (e.g., unrealistic delivery date)
2. **2. Predictable risks:** Those risks that are hypothesized from previous project experience (e.g., past turnover)
3. **3. Unpredictable risks:** Those risks that can and do occur, but are extremely tough to identify in advance.

Principle of Risk Management

1. **Global Perspective:** In this, we review the bigger system description, design, and implementation. We look at the chance and the impact the risk is going to have.
2. **Take a forward-looking view:** Consider the threat which may appear in the future and create future plans for directing the next events.
3. **Open Communication:** This is to allow the free flow of communications between the client and the team members so that they have certainty about the risks.
4. **Integrated management:** In this method risk management is made an integral part of project management.
5. **Continuous process:** In this phase, the risks are tracked continuously throughout the risk management paradigm.

Risk management process

- ✧ Risk identification
  - Identify project, product and business risks;
- ✧ Risk analysis
  - Assess the likelihood and consequences of these risks;
- ✧ Risk planning
  - Draw up plans to avoid or minimise the effects of the risk;
- ✧ Risk monitoring
  - Monitor the risks throughout the project;