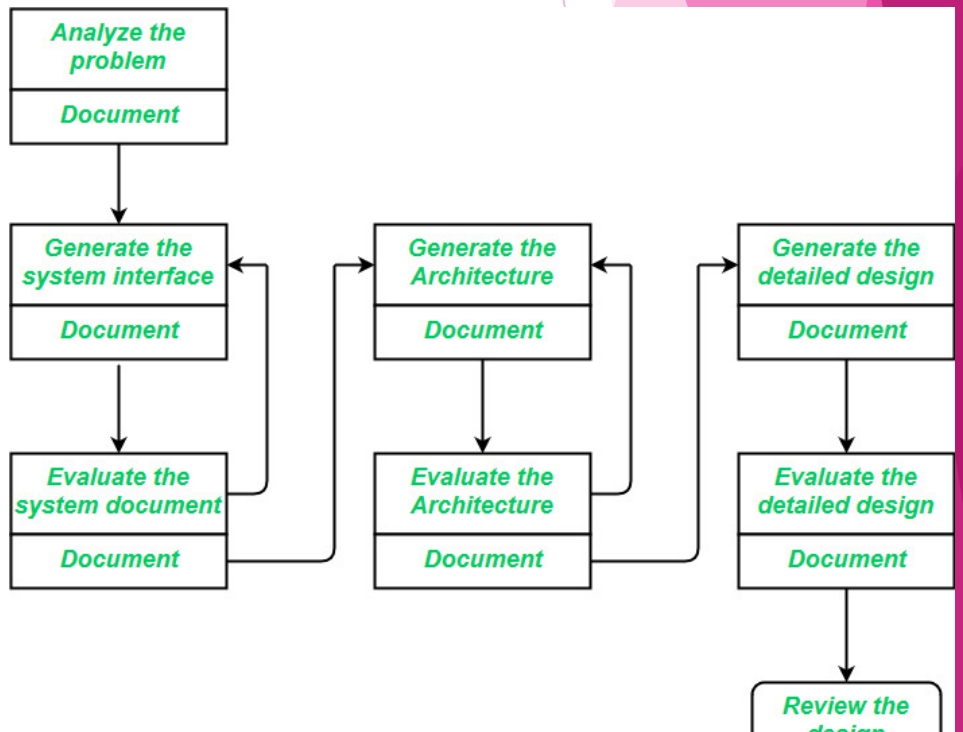
UNIT - 4 : Design Engineering& Architecture, Testing Strategie

Design process and Design quality

- **Elements of a System**
- **Architecture:** This is the conceptual model that defines the structure, behavior, and views of a system. We can use flowcharts to represent and illustrate the architecture.
- **Modules:** These are components that handle one specific task in a system. A combination of the modules makes up the system.
- **Components:** This provides a particular function or group of related functions. They are made up of modules.
- **Interfaces:** This is the shared boundary across which the components of a system exchange information and relate.
- **Data:** This is the management of the information and data flow.
-



- **Interface Design**
- Interface design is the specification of the interaction between a system and its environment. This phase proceeds at a high level of abstraction with respect to the inner workings of the system i.e, during interface design, the internal of the systems are completely ignored, and the system is treated as a black box. Attention is focused on the dialogue between the target system and the users, devices, and other systems with which it interacts. The design problem statement produced during the problem analysis step should identify the people, other systems, and devices which are collectively called agents.
- Interface design should include the following details:
- Precise description of events in the environment, or messages from agents to which the system must respond.
- Precise description of the events or messages that the system must produce.
- Specification of the data, and the formats of the data coming into and going out of the system.
- Specification of the ordering and timing relationships between incoming events or messages, and outgoing events or outputs.
- **Architectural Design**
- Architectural design is the specification of the major components of a system, their responsibilities, properties, interfaces, and the relationships and interactions between them. In architectural design, the overall structure of the system is chosen, but the internal details of major components are ignored. Issues in architectural design includes:
- Gross decomposition of the systems into major components.
- Allocation of functional responsibilities to components.
- Component Interfaces.
- Component scaling and performance properties, resource consumption properties, reliability properties, and so forth.
- Communication and interaction between components.

Detailed Design

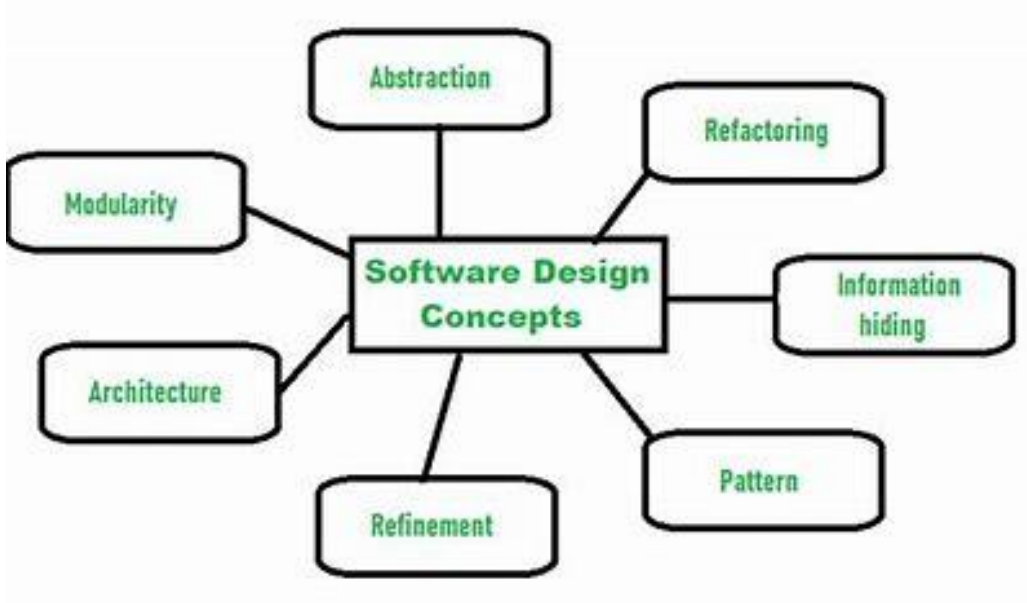
- Detailed design is the specification of the internal elements of all major system components, their properties, relationships, processing, and often their algorithms and the data structures. The detailed design may include:
- Decomposition of major system components into program units.
- Allocation of functional responsibilities to units.
- User interfaces.
- Unit states and state changes.
- Data and control interaction between units.
- Data packaging and implementation, including issues of scope and visibility of program elements.
- Algorithms and data structures.
-

Design concepts

- Software Design is the process of transforming user requirements into a suitable form, which helps the programmer in software coding and implementation. During the software design phase, the design document is produced, based on the customer requirements as documented in the SRS document. Hence, this phase aims to transform the SRS document into a design document.
- The following items are designed and documented during the design phase:
- Different modules are required.
- Control relationships among modules.
- Interface among different modules.
- Data structure among the different modules.
- Algorithms are required to be implemented among the individual modules
- **Objectives of Software Design**
- **Correctness:** A good design should be correct i.e., it should correctly implement all the functionalities of the system.
- **Efficiency:** A good software design should address the resources, time, and cost optimization issues.
- **Flexibility:** A good software design should have the ability to adapt and accommodate changes easily. It includes designing the software in a way, that allows for modifications, enhancements, and scalability without requiring significant rework or causing major disruptions to the existing functionality.
- **Understandability:** A good design should be easily understandable, it should be modular, and all the modules are arranged in layers.
- **Completeness:** The design should have all the components like data structures, modules, external interfaces, etc.
- **Maintainability:** A good software design aims to create a system that is easy to understand, modify, and maintain over time. This involves using modular and well-structured design principles e.g.,(employing appropriate naming conventions and providing clear documentation). Maintainability in Software and design also enables developers to fix bugs, enhance features, and adapt the software to changing requirements without excessive effort or introducing new issues.

Software Design Concepts

- Concepts are defined as a principal idea or invention that comes into our mind or in thought to understand something. The **software design concept** simply means the idea or principle behind the design. It describes how you plan to solve the problem of designing software, and the logic, or thinking behind how you will design software. It allows the software engineer to create the model of the system software or product that is to be developed or built. The software design concept provides a supporting and essential structure or model for developing the right software. There are many concepts of software design and some of them are given below:



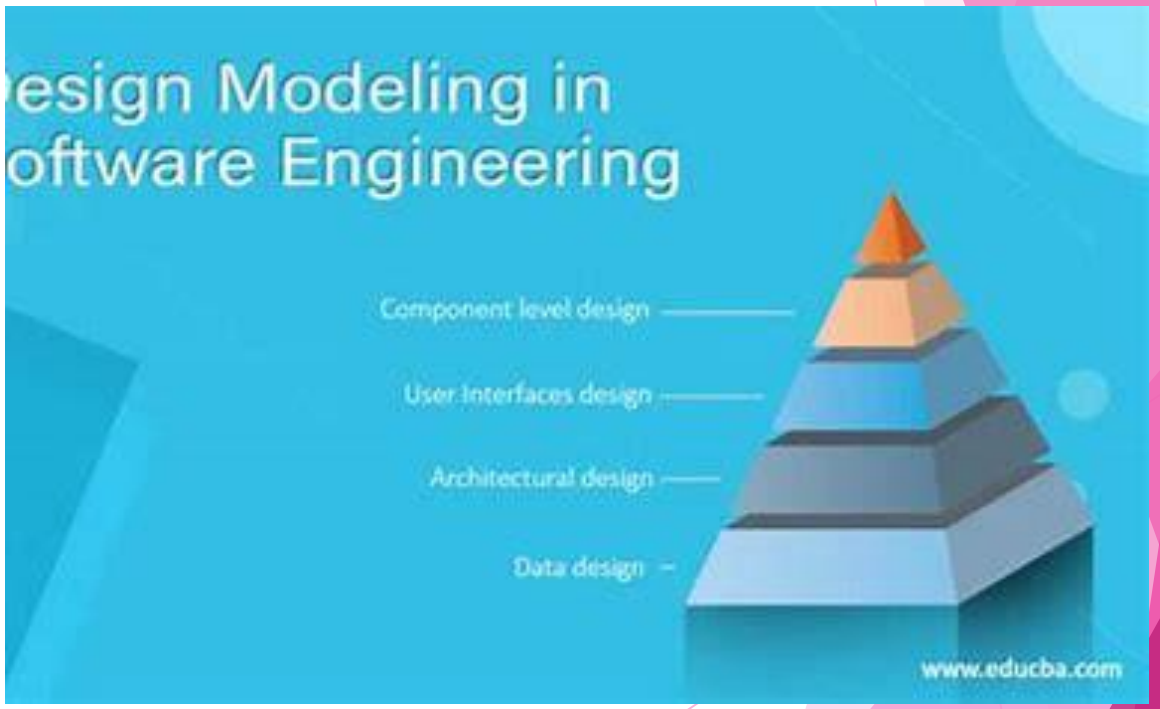
Points to be Considered While Designing Software

- **Abstraction (Hide Irrelevant data):** Abstraction simply means to hide the details to reduce complexity and increase efficiency or quality. Different levels of Abstraction are necessary and must be applied at each stage of the design process so that any error that is present can be removed to increase the efficiency of the software solution and to refine the software solution. The solution should be described in broad ways that cover a wide range of different things at a higher level of abstraction and a more detailed description of a solution of software should be given at the lower level of abstraction.
- **Modularity (subdivide the system):** Modularity simply means dividing the system or project into smaller parts to reduce the complexity of the system or project. In the same way, modularity in design means subdividing a system into smaller parts so that these parts can be created independently and then use these parts in different systems to perform different functions. It is necessary to divide the software into components known as modules because nowadays, there are different software available like Monolithic software that is hard to grasp for software engineers. So, modularity in design has now become a trend and is also important. If the system contains fewer components then it would mean the system is complex which requires a lot of effort (cost) but if we can divide the system into components then the cost would be small.
- **Architecture (design a structure of something):** Architecture simply means a technique to design a structure of something. Architecture in designing software is a concept that focuses on various elements and the data of the structure. These components interact with each other and use the data of the structure in architecture.

- **Refinement (removes impurities):** Refinement simply means to refine something to remove any impurities if present and increase the quality. The refinement concept of software design is a process of developing or presenting the software or system in a detailed manner which means elaborating a system or software. Refinement is very necessary to find out any error if present and then to reduce it.
- **Pattern (a Repeated form):** A pattern simply means a repeated form or design in which the same shape is repeated several times to form a pattern. The pattern in the design process means the repetition of a solution to a common recurring problem within a certain context.
- **Information Hiding (Hide the Information):** Information hiding simply means to hide the information so that it cannot be accessed by an unwanted party. In software design, information hiding is achieved by designing the modules in a manner that the information gathered or contained in one module is hidden and can't be accessed by any other modules.
- **Refactoring (Reconstruct something):** Refactoring simply means reconstructing something in such a way that it does not affect the behavior of any other features. Refactoring in software design means reconstructing the design to reduce complexity and simplify it without impacting the behavior or its functions. Fowler has defined refactoring as "the process of changing a software system in a way that it won't impact the behavior of the design and improves the internal structure".

The design model

Design modeling in software engineering represents the features of the software that helps engineer to develop it effectively, the architecture, the user interface, and the component level detail. Design modeling provides a variety of different views of the system like architecture plan for home or building. Different methods like data-driven, pattern-driven, or object-oriented methods are used for constructing the design model. All these methods use set of design principles for designing a model.



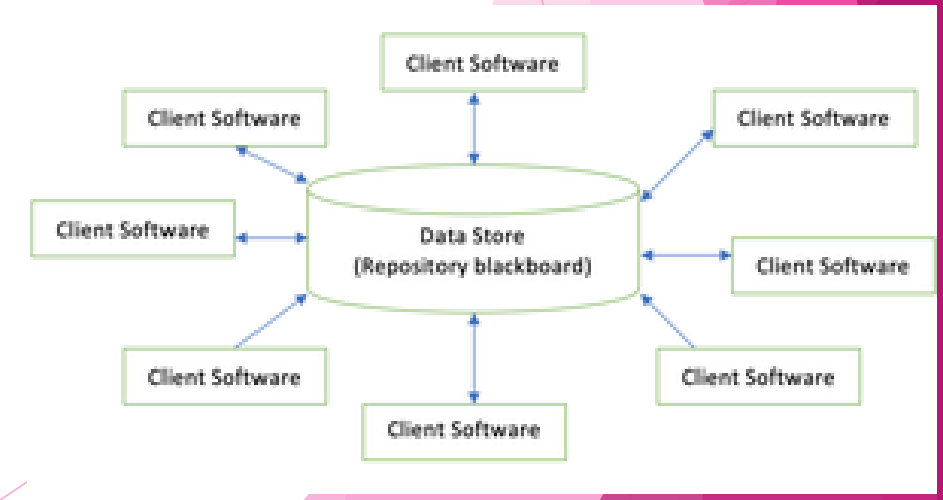
- **Data design:** It represents the data objects and their interrelationship in an entity-relationship diagram. Entity-relationship consists of information required for each entity or data objects as well as it shows the relationship between these objects. It shows the structure of the data in terms of the tables. It shows three type of relationship – One to one, one to many, and many to many. In one to one relation, one entity is connected to another entity. In one many relation, one Entity is connected to more than one entity. un many to many relations one entity is connected to more than one entity as well as other entity also connected with first entity using more than one entity.
- **Architectural design:** It defines the relationship between major structural elements of the software. It is about decomposing the system into interacting components. It is expressed as a block diagram defining an overview of the system structure – features of the components and how these components communicate with each other to share data. It defines the structure and properties of the component that are involved in the system and also the inter-relationship among these components.
- **User Interfaces design:** It represents how the Software communicates with the user i.e. the behavior of the system. It refers to the product where user interact with controls or displays of the product. For example, Military, vehicles, aircraft, audio equipment, computer peripherals are the areas where user interface design is implemented. UI design becomes efficient only after performing usability testing. This is done to test what works and what does not work as expected. Only after making the repair, the product is said to have an optimized interface.
- **Component level design:** It transforms the structural elements of the software architecture into a procedural description of software components. It is a perfect way to share a large amount of data. Components need not be concerned with how data is managed at a centralized level i.e. components need not worry about issues like backup and security of the data.

Principles of Design Model

- **Design must be traceable to the analysis model:**
- Analysis model represents the information, functions, and behavior of the system. Design model translates all these things into architecture – a set of subsystems that implement major functions and a set of component kevel design that are the realization of Analysis classes. This implies that design model must be traceable to the analysis model.
- **Always consider architecture of the system to be built:**
- Software architecture is the skeleton of the system to be built. It affects interfaces, data structures, behavior, program control flow, the manner in which testing is conducted, maintainability of the resultant system, and much more.
- **Focus on the design of the data:**
- Data design encompasses the manner in which the data objects are realized within the design. It helps to simplify the program flow, makes the design and implementation of the software components easier, and makes overall processing more efficient.
- **User interfaces should consider the user first:**
- The user interface is the main thing of any software. No matter how good its internal functions are or how well designed its architecture is but if the user interface is poor and end-users don’t feel ease to handle the software then it leads to the opinion that the software is bad.

Creating an architectural design

- The software needs an architectural design to represent the design of the software. IEEE defines architectural design as “the process of defining a collection of hardware and software components and their interfaces to establish the framework for the development of a computer system.” The software that is built for computer-based systems can exhibit one of these many architectural styles.
- **1] Data centered architectures:**
- A data store will reside at the center of this architecture and is accessed frequently by the other components that update, add, delete, or modify the data present within the store.
- The figure illustrates a typical data-centered style. The client software accesses a central repository. Variations of this approach are used to transform the repository into a blackboard when data related to the client or data of interest for the client change the notifications to client software.
- This data-centered architecture will promote integrability. This means that the existing components can be changed and new client components can be added to the architecture without the permission or concern of other clients.
- Data can be passed among clients using the blackboard mechanism.
- **Advantages of Data centered architecture:**
- Repository of data is independent of clients
- Client work independent of each other
- It may be simple to add additional clients.
- Modification can be very easy



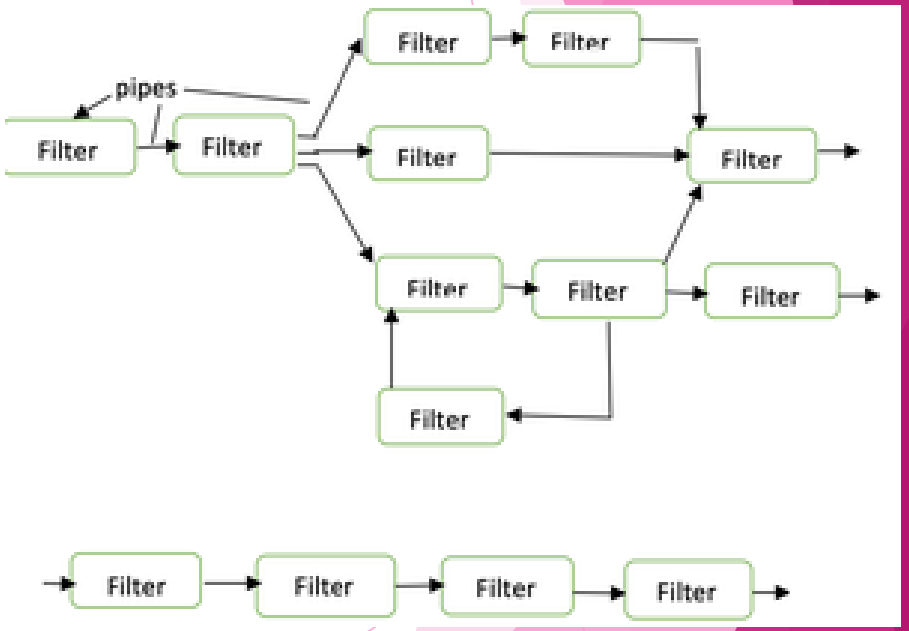
- **] Data flow architectures:**
- This kind of architecture is used when input data is transformed into output data through a series of computational manipulative components.
- The figure represents pipe-and-filter architecture since it uses both pipe and filter and it has a set of components called filters connected by lines.
- Pipes are used to transmitting data from one component to the next.
- Each filter will work independently and is designed to take data input of a certain form and produces data output to the next filter of a specified form. The filters don't require any knowledge of the working of neighboring filters.
- If the data flow degenerates into a single line of transforms, then it is termed as batch sequential. This structure accepts the batch of data and then applies a series of sequential components to transform it.
- **Advantages of Data Flow architecture:**
- It encourages upkeep, repurposing, and modification.
- With this design, concurrent execution is supported.
- **Disadvantage of Data Flow architecture:**
- It frequently degenerates to batch sequential system
- Data flow architecture does not allow applications that require greater user engagement.
- It is not easy to coordinate two different but related streams
-

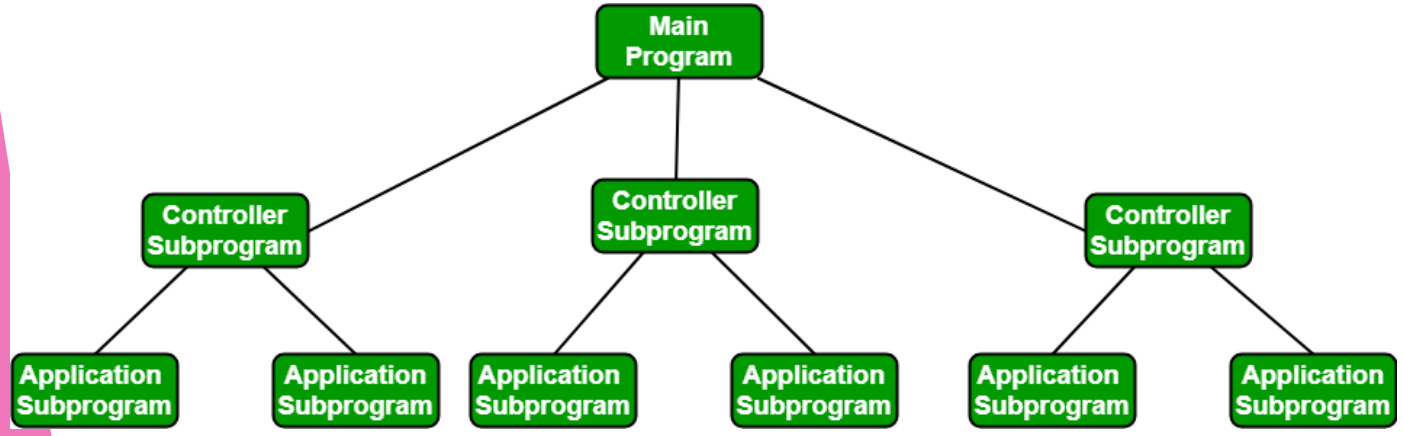
3] Call and Return architectures

It is used to create a program that is easy to scale and modify. Many sub-styles exist within this category. Two of them are explained below.

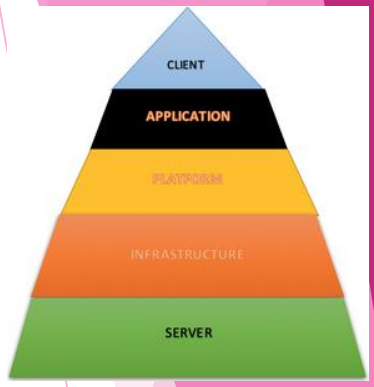
•**Remote procedure call architecture:** This components is used to present in a main program or sub program architecture distributed among multiple computers on a network.

•**Main program or Subprogram architectures:** The main program structure decomposes into number of subprograms or function into a control hierarchy. Main program contains number of subprograms that can invoke other components.





- **] Object Oriented architecture**
- The components of a system encapsulate data and the operations that must be applied to manipulate the data. The coordination and communication between the components are established via the message passing.
- **Characteristics of Object Oriented architecture:**
- Object protect the system's integrity.
- An object is unaware of the depiction of other items.
- **Advantage of Object Oriented architecture:**
- It enables the designer to separate a challenge into a collection of autonomous objects.
- Other objects are aware of the implementation details of the object, allowing changes to be made without having an impact on other objects.
- **5] Layered architecture**
- A number of different layers are defined with each layer performing a well-defined set of operations. Each layer will do some operations that becomes closer to machine instruction set progressively.
- At the outer layer, components will receive the user interface operations and at the inner layers, components will perform the operating system interfacing(communication and coordination with OS)
- Intermediate layers to utility services and application software functions.



software architecture

- Software Architecture defines fundamental organization of a system and more simply defines a structured solution. It defines how components of a software system are assembled, their relationship and communication between them. It serves as a blueprint for software application and development basis for developer team.
- Software architecture defines a list of things which results in making many things easier in the software development process.
- A software architecture defines structure of a system.
- A software architecture defines behavior of a system.
- A software architecture defines component relationship.
- A software architecture defines communication structure.
- A software architecture balances stakeholder’s needs.
- A software architecture influences team structure.
- A software architecture focuses on significant elements.
- A software architecture captures early design decisions.

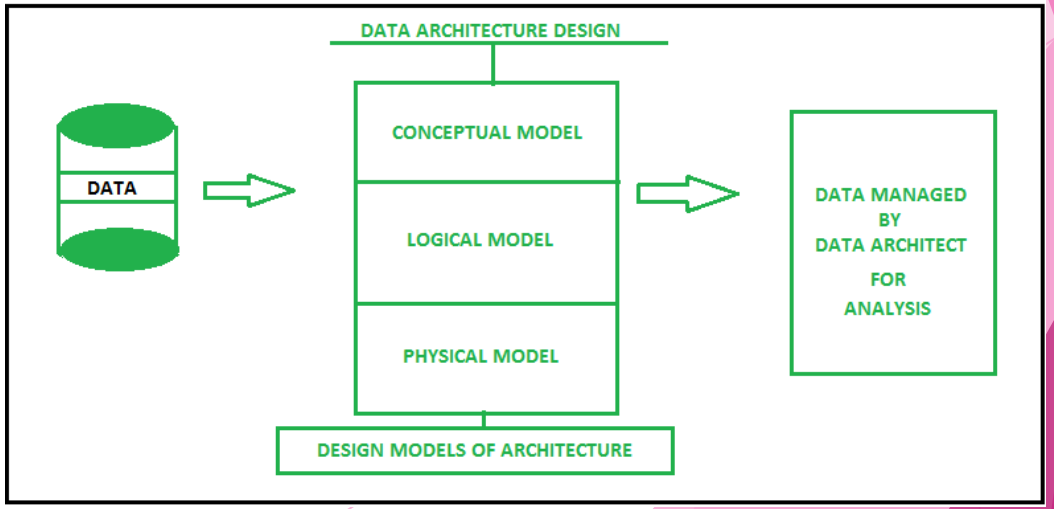
Characteristics of Software Architecture :

- Architects separate architecture characteristics into broad categories depending upon operation, rarely appearing requirements, structure etc. Below some important characteristics which are commonly considered are explained.
- **Operational Architecture Characteristics :**
  - Availability
  - Performance
  - Reliability
  - Low fault tolerance
  - Scalability
- **Structural Architecture Characteristics :**
  - Configurability
  - Extensibility
  - Supportability
  - Portability
  - Maintainability
- **Cross-Cutting Architecture Characteristics :**
  - Accessibility
  - Security
  - Usability
  - Privacy
  - Feasibility

Data design

- **Data design** is the first design activity, which results in less complex, modular and efficient program structure. The information domain model developed during analysis phase is transformed into data structures needed for implementing the software. The data objects, attributes, and relationships depicted in entity relationship diagrams and the information stored in data dictionary provide a base for data design activity. During the data design process, data types are specified along with the integrity rules required for the data. For specifying and designing efficient data structures, some principles should be followed. These principles are listed below.
- The data structures needed for implementing the software as well-as the operations that can be applied on them should be identified.
- A data dictionary should be developed to depict how different data objects interact with each other and what constraints are to be imposed on the elements of data structure.
- Stepwise refinement should be used in data design process and detailed design decisions should be made later in the process.
- Only those modules that need to access data stored in a data structure directly should be aware of the representation of the data structure.
- A library containing the set of useful data structures along with the operations that can be performed on them should be maintained.
- Language used for developing the system should support abstract data types.

- The structure of data can be viewed at three levels, namely, *program* component level, application level, and business level. At the **program component level,** the design of data structures and the algorithms required to manipulate them is necessary, if high-quality software is desired. At the **application level,** it is crucial to convert the data model into a database so that the specific business objectives of a system could be achieved. At the **business level,** the collection of information stored in different databases should be reorganized into data warehouse, which enables data mining that has an influential impact on the business.
- **You'll also like:**
- Principles of Software Design & Concepts in Software Engineering
- Software Design Reviews in Software Engineering
- Architectural Design in Software Engineering
- Component-Level Design in software engineering
- Software Engineering – What is Software Engineering? Write Basic Objective and Need for Software Engineering



- **Data architecture design** is important for creating a vision of interactions occurring between data systems, like for example if data architect wants to implement data integration, so it will need interaction between two systems and by using data architecture the visionary model of data interaction during the process can be achieved.
- Data architecture also describes the type of data structures applied to manage data and it provides an easy way for data preprocessing. The data architecture is formed by dividing into three essential models and then are combined :
- **Conceptual model –**
  It is a business model which uses Entity Relationship (ER) model for relation between entities and their attributes.
- **Logical model –**
  It is a model where problems are represented in the form of logic such as rows and column of data, classes, xml tags and other DBMS techniques.
- **Physical model –**
  Physical models holds the database design like which type of database technology will be suitable for architecture.
- A data architect is responsible for all the design, creation, manage, deployment of data architecture and defines how data is to be stored and retrieved, other decisions are made by internal bodies.
- **Factors that influence Data Architecture :**
  Few influences that can have an effect on data architecture are business policies, business requirements, Technology used, economics, and data processing needs.
- **Business requirements –**
  These include factors such as the expansion of business, the performance of the system access, data management, transaction management, making use of raw data by converting them into image files and records, and then storing in data warehouses. Data warehouses are the main aspects of storing transactions in business.

- **Business policies -**
  The policies are rules that are useful for describing the way of processing data. These policies are made by internal organizational bodies and other government agencies.
- **Technology in use -**
  This includes using the example of previously completed data architecture design and also using existing licensed software purchases, database technology.
- **Business economics -**
  The economical factors such as business growth and loss, interest rates, loans, condition of the market, and the overall cost will also have an effect on design architecture.
- **Data processing needs -**
  These include factors such as mining of the data, large continuous transactions, database management, and other data preprocessing needs.
- **Data Management :**
- Data management is the process of managing tasks like extracting data, storing data, transferring data, processing data, and then securing data with low-cost consumption.
- Main motive of data management is to manage and safeguard the people's and organization data in an optimal way so that they can easily create, access, delete, and update the data.
- Because data management is an essential process in each and every enterprise growth, without which the policies and decisions can't be made for business advancement. The better the data management the better productivity in business.

Architectural styles and patterns

- Software architecture is like the blueprint for building software, showing how different parts fit together and interact. It helps the development team understand how to build the software according to customer requirements. There are many ways to organize these parts, called software architecture patterns. These patterns have been tested and proven to solve different problems by arranging components in specific ways. This article focuses on discussing Software Architecture Patterns in detail.
- **What is a Software Architecture?**
- Software Architecture is a high-level structure of the software system that includes the set of rules, patterns, and guidelines that dictate the organization, interactions, and the component relationships. It serves as blueprint ensuring that the system meets its requirements and it is maintainable and scalable.
- **Modularity:** Software Architecture divides system into interchangeable components that can be developed, tested, and maintained independently.
- **Encapsulation:** It helps to hide the details of the components, exposing only the necessary information, thus reducing the complexity of the system.
- **Security:** It helps to incorporate measures to protect the system against the unauthorized access.
- **Documentation:** Provides clear documentation of the system architecture, thus facilitating communication and better understanding of the system.
- **Performance:** Software architecture helps to make sure that the system meets the required performance metrics such as resource utilization, throughput, etc.

Types of Software Architecture Patterns

- **1. Layered Architecture Pattern**
- As the name suggests, components(code) in this pattern are separated into layers of subtasks and they are arranged one above another. Each layer has unique tasks to do and all the layers are independent of one another. Since each layer is independent, one can modify the code inside a layer without affecting others. It is the most commonly used pattern for designing the majority of software. This layer is also known as 'N-tier architecture'. Basically, this pattern has 4 layers.
- **Presentation layer:** The user interface layer where we see and enter data into an application.)
- **Business layer:** This layer is responsible for executing business logic as per the request.)
- **Application layer:** This layer acts as a medium for communication between the 'presentation layer' and 'data layer'.
- **Data layer:** This layer has a database for managing data.)
- **Advantages:**
- **Scalability:** Individual layers in the architecture can be scaled independently to meet performance needs.
- **Flexibility:** Different technologies can be used within each layer without affecting others.
- **Maintainability:** Changes in one layer do not necessarily impact other layers, thus simplifying the maintenance.
- **Disadvantages:**
- **Complexity:** Adding more layers to the architecture can make the system more complex and difficult to manage.
- **Performance Overhead:** Multiple layers can introduce latency due to additional communication between the layers.
- **Strict Layer Separation:** Strict layer separation can sometimes lead to inefficiencies and increased development effort.

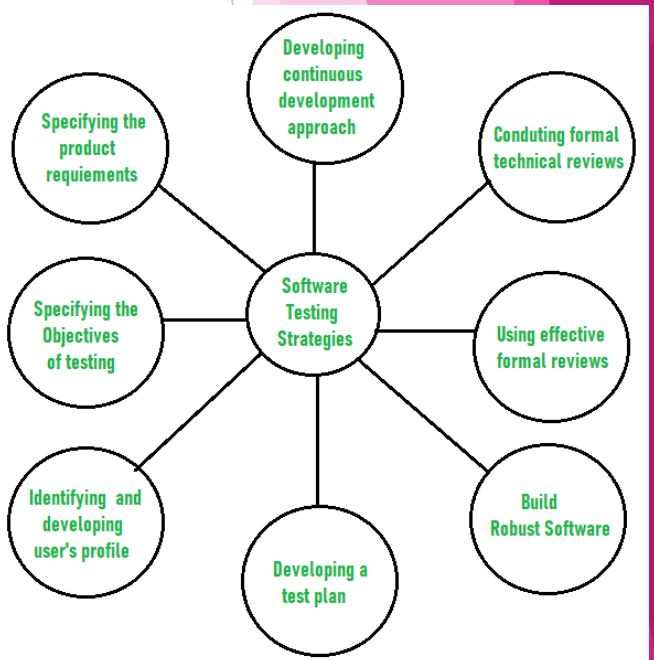
- **2. Client-Server Architecture Pattern**
- The client-server pattern has two major entities. They are a server and multiple clients. Here the server has resources(data, files or services) and a client requests the server for a particular resource. Then the server processes the request and responds back accordingly.
- **Advantages:**
- **Centralized Management:** Servers can centrally manage resources, data, and security policies, thus simplifying the maintenance.
- **Scalability:** Servers can be scaled up to handle increased client requests.
- **Security:** Security measures such as access controls, data encryption can be implemented in a better way due to centralized controls.
- **Disadvantages:**
- **Single Point of Failure:** Due to centralized server, if server fails clients lose access to services, leading loss of productivity.
- **Costly:** Setting up and maintaining servers can be expensive due to hardware, software, and administrative costs.
- **Complexity:** Designing and managing a client-server architecture can be complex. **Use Cases:**
- Web Applications like Amazon.
- Email Services like Gmail, Outlook.
- File Sharing Services like Dropbox, Google Drive.
- Media Streaming Services like Netflix.
- Education Platforms like Moodle.

- **3. Event-Driven Architecture Pattern**
- Event-Driven Architecture is an agile approach in which services (operations) of the software are triggered by events. When a user takes action in the application built using the EDA approach, a state change happens and a reaction is generated that is called an event.
- **Example:**
- A new user fills the signup form and clicks the signup button on Facebook and then a FB account is created for him, which is an event.
- **Advantages:**
- **Scalability:** System can scale horizontally by adding more consumers.
- **Real-time Processing:** This enables real-time processing and immediate response to events.
- **Flexibility:** New event consumers can be added without modifying existing components.
- **Disadvantages:**
- **Complexity:** The architecture can be complex to design, implement, and debug.
- **Complex Testing:** Testing event-driven systems can be complicated compared to synchronous systems.
- **Reliability:** Ensuring reliability requires additional mechanisms to handle failed events.
- **Use Cases:**
- Real-Time Analytics like stock market analysis systems.
- IoT Applications like smart home systems.

strategic approach to software testing

- Software testing is the process of evaluating a software application to identify if it meets specified requirements and to identify any defects. The following are common testing strategies:
- **Black box testing** – Tests the functionality of the software without looking at the internal code structure.
- **White box testing** – Tests the internal code structure and logic of the software.
- **Unit testing** – Tests individual units or components of the software to ensure they are functioning as intended.
- **Integration testing** – Tests the integration of different components of the software to ensure they work together as a system.
- **Functional testing** – Tests the functional requirements of the software to ensure they are met.
- **System testing** – Tests the complete software system to ensure it meets the specified requirements.
- **Acceptance testing** – Tests the software to ensure it meets the customer's or end-user's expectations.
- **Regression testing** – Tests the software after changes or modifications have been made to ensure the changes have not introduced new defects.
- **Performance testing** – Tests the software to determine its performance characteristics such as speed, scalability, and stability.
- **Security testing** – Tests the software to identify vulnerabilities and ensure it meets security requirements.

- Software Testing is a type of investigation to find out if there is any default or error present in the software so that the errors can be reduced or removed to increase the quality of the software and to check whether it fulfills the specifies requirements or not.
  According to Glen Myers, software testing has the following objectives:
- The process of investigating and checking a program to find whether there is an error or not and does it fulfill the requirements or not is called testing.
- When the number of errors found during the testing is high, it indicates that the testing was good and is a sign of good test case.
- Finding an unknown error that wasn't discovered yet is a sign of a successful and a good test case.
- The main objective of software testing is to design the tests in such a way that it systematically finds different types of errors without taking much time and effort so that less time is required for the development of the software. The overall strategy for testing software includes:



- **Advantages of software testing:**
- Improves software quality and reliability – Testing helps to identify and fix defects early in the development process, reducing the risk of failure or unexpected behavior in the final product.
- Enhances user experience – Testing helps to identify usability issues and improve the overall user experience.
- Increases confidence – By testing the software, developers and stakeholders can have confidence that the software meets the requirements and works as intended.
- Facilitates maintenance – By identifying and fixing defects early, testing makes it easier to maintain and update the software.
- Reduces costs – Finding and fixing defects early in the development process is less expensive than fixing them later in the life cycle.
- **Disadvantages of software testing:**
- Time-consuming – Testing can take a significant amount of time, particularly if thorough testing is performed.
- Resource-intensive – Testing requires specialized skills and resources, which can be expensive.
- Limited coverage – Testing can only reveal defects that are present in the test cases, and it is possible for defects to be missed.
- Unpredictable results – The outcome of testing is not always predictable, and defects can be hard to replicate and fix.
- Delays in delivery – Testing can delay the delivery of the software if testing takes longer than expected or if significant defects are identified.

- **Before testing starts, it's necessary to identify and specify the requirements of the product in a quantifiable manner.** Different characteristics quality of the software is there such as maintainability that means the ability to update and modify, the probability that means to find and estimate any risk, and usability that means how it can easily be used by the customers or end-users. All these characteristic qualities should be specified in a particular order to obtain clear test results without any error.
- **Specifying the objectives of testing in a clear and detailed manner.** Several objectives of testing are there such as effectiveness that means how effectively the software can achieve the target, any failure that means inability to fulfill the requirements and perform functions, and the cost of defects or errors that mean the cost required to fix the error. All these objectives should be clearly mentioned in the test plan.
- **For the software, identifying the user's category and developing a profile for each user.** Use cases describe the interactions and communication among different classes of users and the system to achieve the target. So as to identify the actual requirement of the users and then testing the actual use of the product.
- **Developing a test plan to give value and focus on rapid-cycle testing.** Rapid Cycle Testing is a type of test that improves quality by identifying and measuring the any changes that need to be required for improving the process of software. Therefore, a test plan is an important and effective document that helps the tester to perform rapid cycle testing. **Robust software is developed that is designed to test itself.** The software should be capable of detecting or identifying different classes of errors. Moreover, software design should allow automated and regression testing which tests the software to find out if there is any adverse or side effect on the features of software due to any change in code or program.
- **Before testing, using effective formal reviews as a filter.** Formal technical reviews is technique to identify the errors that are not discovered yet. The effective technical reviews conducted before testing reduces a significant amount of testing efforts and time duration required for testing software so that the overall development time of software is reduced.
- **Conduct formal technical reviews to evaluate the nature, quality or ability of the test strategy and test cases.** The formal technical review helps in detecting any unfilled gap in the testing approach. Hence, it is necessary to evaluate the ability and quality of the test strategy and test cases by technical reviewers to improve the quality of software.
- **For the testing process, developing a approach for the continuous development.** As a part of a statistical process control approach, a test strategy that is already measured should be used for software testing to measure and control the quality during the development of software.

Test strategies for conventional Software

- 1. Understanding Conventional Software Testing
- A. **Definition:** Conventional software testing refers to the testing process applied to traditional software applications that follow well-defined specifications and requirements.
- B. **Test Objectives:** The primary objectives of conventional software testing are to identify defects, ensure functionality, and verify adherence to requirements.
- **2. Test Strategy vs. Test Plan**
- A. **Test Strategy:** A test strategy outlines the overall approach, scope, and resources for software testing.
- B. **Test Plan:** A test plan provides detailed information on individual test cases, test scenarios, and execution schedules
- **3. Popular Test Strategies for Conventional Software**
- A. **1. Waterfall Model Testing:**
- **Description:** Waterfall model follows a linear and sequential development approach, and testing is carried out after development.
- **Advantages:** Clear requirements and well-defined phases make testing more systematic.
- **Challenges:** Late defect identification and minimal scope for accommodating changes during testing.
- B. **2. Agile Testing:**
- **Description:** Agile testing aligns with iterative development and frequent testing to enable continuous integration and delivery.
- **Advantages:** Early and continuous testing ensures faster feedback and adaptability to changing requirements.

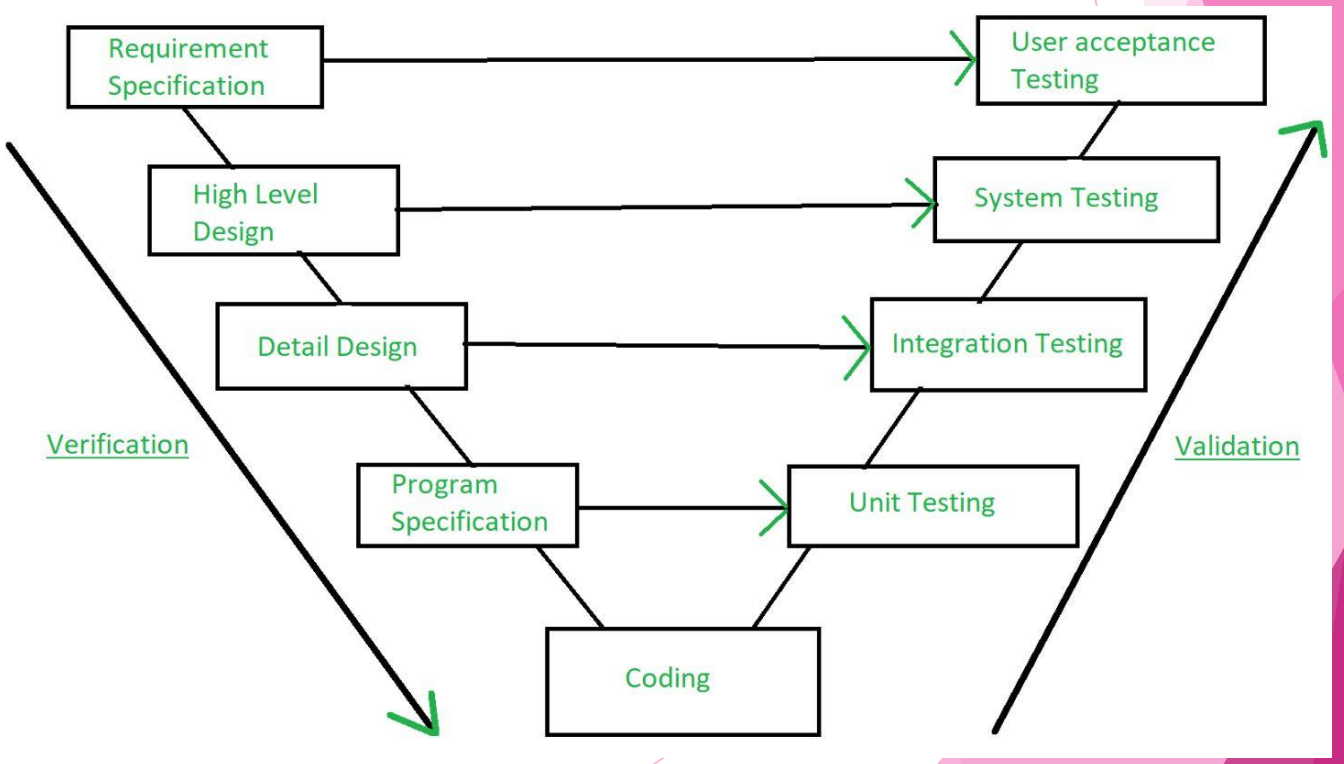
- C. **3. V-Model Testing:**
- **Description:** V-Model correlates development phases with testing phases, emphasizing validation and verification activities.
- **Advantages:** Well-structured and comprehensive testing at every stage of development.
- **Challenges:** Prone to delays if defects are identified late in the process.
- **4. Black Box and White Box Testing in Conventional Software**
- A. **Black Box Testing:**
- **Objective:** Evaluates software functionality without considering its internal structure.
- **Advantages:** Focuses on user perspective, independent of code implementation.
- **Challenges:** Limited visibility into the internal workings of the software.
- B. **White Box Testing:**
- **Objective:** Examines the internal logic and code structure of the software.
- **Advantages:** Thorough code coverage and precise defect identification.
- **Challenges:** Requires knowledge of code
- **5. Test Automation in Conventional Software Testing**
- A. **Automated Test Suites:** Create automated test suites to execute repetitive test cases efficiently.
- B. **Regression Testing:** Automate regression testing to validate that software changes do not impact existing functionalities.
- C. **Performance Testing:** Automate performance testing to assess system responsiveness and scalability.
- **6. Best Practices for Conventional Software Testing**
- A. **Early Testing:** Begin testing early in the development process to identify and fix defects at the source.
- B. **Collaborative Approach:** Foster collaboration between development and testing teams for better communication and issue resolution.

Validation testing

- **Verification and Validation** is the process of investigating whether a software system satisfies specifications and standards and fulfills the required purpose. **Barry Boehm** described verification and validation as the following:
- ***Verification****: Are we building the product right?*
- ***Validation****: Are we building the right product?*
- **Verification**
- Verification is the process of checking that 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 simply known as **Static Testing.**
- **Static Testing**
- Verification Testing is known as Static Testing and it can be simply termed as checking whether we are developing the right product or not and also whether our software is fulfilling the customer's requirement or not. Here are some of the activities that are involved in verification.
- Inspections
- Reviews
- Walkthroughs
- Desk-checking

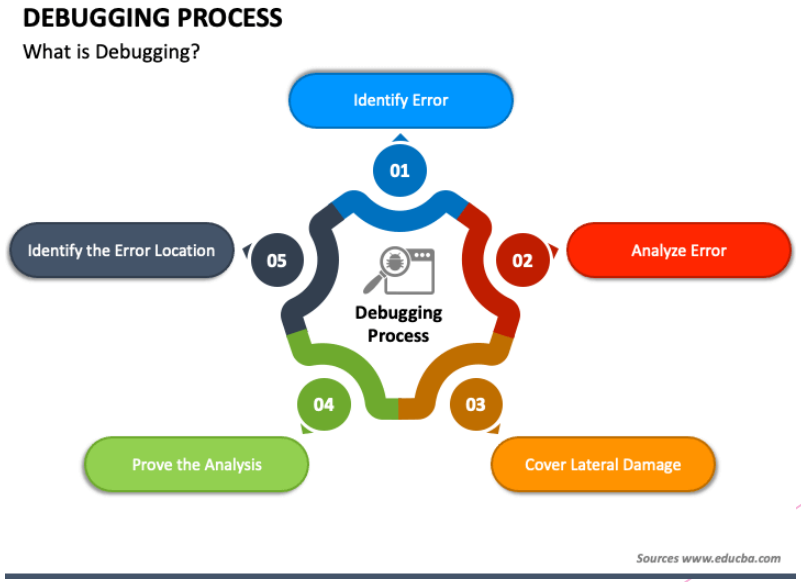
Validation

- 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 the product i.e. it checks what we are developing is the right product. it is a validation of actual and expected products. Validation is simply known as **Dynamic Testing**.
- **Dynamic Testing**
- Validation Testing is known as Dynamic Testing in which we examine whether we have developed the product right or not and also about the business needs of the client. Here are some of the activities that are involved in Validation.
- Black Box Testing
- White Box Testing
- Unit Testing
- Integration Testing



The art of debugging.

- **Debugging** in **Software Engineering** is the process of identifying and resolving **errors** or **bugs** in a software system. It's a critical aspect of software development, ensuring **quality**, **performance**, and **user satisfaction**. Despite being time-consuming, effective **debugging** is essential for reliable and competitive software products.
- **What is Debugging?**
- In the context of software engineering, debugging is the process of fixing a bug in the software. When there's a problem with software, programmers analyze the code to figure out why things aren't working correctly. They use different debugging tools to carefully go through the code, step by step, find the issue, and make the necessary corrections.



- **Step 1: Reproduce the Bug**
- To start, you need to **recreate the conditions** that caused the bug. This means making the error happen again so you can see it firsthand.
- Seeing the bug in action helps you understand the problem better and gather important details for fixing it.
- **Step 2: Locate the Bug**
- Next, **find where the bug is in your code**. This involves looking closely at your code and checking any error messages or logs.
- Developers often use debugging tools to help with this step.
- **Step 3: Identify the Root Cause**
- Now, figure out **why the bug happened**. Examine the logic and flow of your code and see how different parts interact under the conditions that caused the bug.
- This helps you understand what went wrong.
- **Step 4: Fix the Bug**
- Once you know the cause, **fix the code**. This involves making changes and then testing the program to ensure the bug is gone.
- Sometimes, you might need to try several times, as initial fixes might not work or could create new issues.
- Using a version control system helps track changes and undo any that don't solve the problem.
- **Step 5: Test the Fix**
- After fixing the bug, **run tests** to ensure everything works correctly. These tests include:
- **Unit Tests:** Check the specific part of the code that was changed.
- **Integration Tests:** Verify the entire module where the bug was found.
- **System Tests:** Test the whole system to ensure overall functionality.
- **Regression Tests:** Make sure the fix didn't cause any new problems elsewhere in the application.
- **Step 6: Document the Process**
- Finally, **record what you did**. Write down what caused the bug, how you fixed it, and any other important details.
- This documentation is helpful if similar issues occur in the future.

Why is debugging important?

- Fixing mistakes in computer programming, known as bugs or errors, is necessary because programming deals with abstract ideas and concepts. Computers understand machine language, but we use programming languages to make it easier for people to talk to computers. Software has many layers of abstraction, meaning different parts must work together for an application to function properly. When errors happen, finding and fixing them can be tricky. That's where debugging tools and strategies come in handy. They help solve problems faster, making developers more efficient. This not only improves the quality of the software but also makes the experience better for the people using it. In simple terms, debugging is important because it makes sure the software works well and people have a good time using it.
- **Debugging Approaches/Strategies**
- **Brute Force:** Study the system for a longer duration to understand the system. It helps the debugger to construct different representations of systems to be debugged depending on the need. A study of the system is also done actively to find recent changes made to the software.
- **Backtracking:** Backward analysis of the problem which involves tracing the program backward from the location of the failure message to identify the region of faulty code. A detailed study of the region is conducted to find the cause of defects.
- **Forward analysis** of the program involves tracing the program forwards using breakpoints or print statements at different points in the program and studying the results. The region where the wrong outputs are obtained is the region that needs to be focused on to find the defect.
- **Using** A debugging **experience** with the software debug the software with similar problems in nature. The success of this approach depends on the expertise of the debugger.
- **Cause elimination:** it introduces the concept of binary partitioning. Data related to the error occurrence are organized to isolate potential causes.
- **Static analysis:** Analyzing the code without executing it to identify potential bugs or errors. This approach involves analyzing code syntax, data flow, and control flow.
- **Dynamic analysis:** Executing the code and analyzing its behavior at runtime to identify errors or bugs. This approach involves techniques like runtime debugging and profiling.