Department of Computer Science and Engineering

Regulation 2021

III Year – VI Semester

CCS356 OBJECT ORIENTED SOFTWARE ENGINEERING

UNIT I

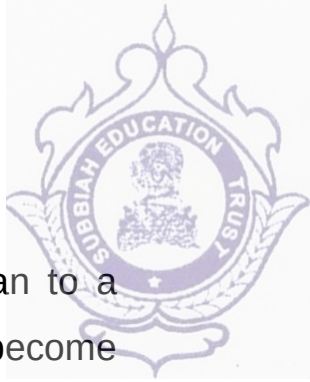
INTRODUCTION TO SOFTWARE ENGINEERING

Software Engineering is a discipline in which theories, methods and tools are applied to develop professional software product**.** It is a systematic and organized approach.

The **software** is a collection of integrated programs. **Engineering** is the application of scientific and practical knowledge to invent, design, build, maintain **and** improve frameworks, processes, etc**.**

Need of Software Engineering

The necessity of software engineering appears because of a higher rate of progress in user requirements and the environment on which the program is working.



Huge Programming: It is simpler to manufacture a wall than to a house or building, similarly, as the measure of programming become extensive engineering has to step to give it a scientific process.

- **Adaptability:** If the software procedure were not based on scientific and engineering ideas, it would be simpler to re-create new software than to scale an existing one.
- **Cost:** the cost of programming remains high if the proper process is not adapted.
- **Dynamic Nature:** If the quality of the software is continually changing, new upgrades need to be done in the existing one.
- **Quality Management:** Better procedure of software development provides a better and quality software product.

The importance of Software engineering is as follows

- Reduces complexity
- To minimize software cost
- To decrease time
- Handling big projects
- Reliable software
- Effectiveness

Software Processes

The term **software** specifies to the set of computer programs, procedures and associated documents (Flowcharts, manuals, etc.) that describe the program and how they are to be used.

A software process is the set of activities and associated outcome that produce a software product. Software engineers mostly carry out these activities. These are four key process activities, which are common to all software processes. These activities are:

1. **Software specifications:** The functionality of the software and constraints on its operation must be defined.
2. **Software development:** The software to meet the requirement must be produced.
3. **Software validation:** The software must be validated to ensure that it does what the customer wants.

1. **Software evolution:** The software must evolve to meet changing client needs.
2. **A workflow model:** This shows the series of activities in the process along with their inputs, outputs and dependencies. The activities in this model perform human actions.
3. **2. A dataflow or activity model:** This represents the process as a set of activities, each of which carries out some data transformations. It shows how the input to the process, such as a specification is converted to an output such as a design. The activities here may be at a lower level than activities in a workflow model. They may perform transformations carried out by people or by computers.
4. **3. A role/action model:** This means the roles of the people involved in the software process and the activities for which they are responsible.

Software Crisis

1. **Size:** Software is becoming more expensive and more complex with the growing complexity and expectation out of software.
2. For example, the code in the consumer product is doubling every couple of years.
3. **Quality:** Many software products have poor quality, i.e., the software products defects after putting into use due to ineffective testing technique. For example, Software testing typically finds 25 errors per 1000 lines of code.
4. **Cost:** Software development is costly i.e. in terms of time taken to develop and the money involved. For example, Development of the FAA's Advanced Automation System cost over $700 per lines of code.
5. **Delayed Delivery:** Very often the software takes longer than the estimated time to develop, which in turn leads to cost shooting up. For example, one in four large-scale development projects are never completed.

PERSPECTIVE PROCESS MODELS

The process model can be defined as the abstract representation of process. It is also known as software development life cycle model or software paradigm.

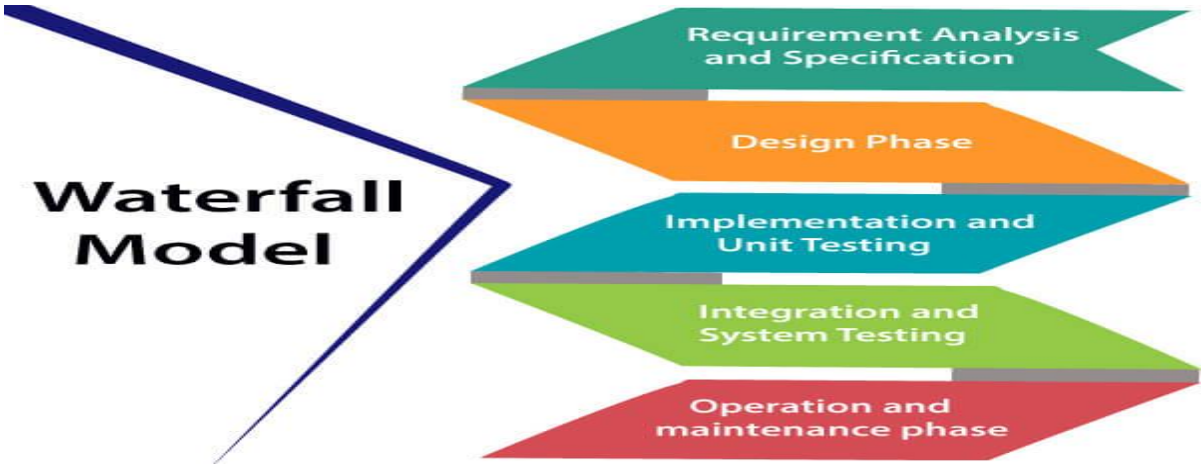
Various activities are carried out in some sequence to make desired software product.every process model consists of definite entry and exit criteria for each process.

WATERFALL MODEL

Winston Royce introduced the Waterfall Model in 1970.This model has five phases: Requirements analysis and specification, design, implementation, and unit testing, integration and system testing, and operation and maintenance.

The developer must complete every phase before the next phase begins. This model is named "**Waterfall Model**", because its diagrammatic representation resembles a cascade of waterfalls.

1. **Requirements analysis and specification phase:** The aim of this phase is to understand the exact requirements of the customer and to document them properly. Both the customer and the software developer work together so as to document all the functions, performance, and interfacing requirement of the software. It describes the "what" of the system to be produced and not "how."



2. Design Phase: This phase aims to transform the requirements gathered in the SRS into a suitable form which permits further coding in a programming language. It defines the overall software architecture together with high level and detailed design. All this work is documented as a Software Design Document (SDD).

3. Implementation and unit testing: During this phase, design is implemented. If the SDD is complete, the implementation or coding phase proceeds smoothly, because all the information needed by software developers is contained in the SDD.

During testing, the code is thoroughly examined and modified. Small modules are tested in isolation initially. After that these modules are tested by writing some overhead code to check the interaction between these modules and the flow of intermediate output.

4. Integration and System Testing: This phase is highly crucial as the quality of the end product is determined by the effectiveness of the testing carried out. The better output will lead to satisfied customers, lower maintenance costs, and accurate results. Unit testing determines the efficiency of individual modules. However, in this phase, the modules are tested for their interactions with each other and with the system.

5. Operation and maintenance phase: Maintenance is the task performed by every user once the software has been delivered to the customer, installed, and operational.

Advantages of Waterfall model

- This model is simple to implement also the number of resources that are required for it is minimal.
- The requirements are simple and explicitly declared; The start and end points for each phase is fixed, which makes it easy to cover progress.
- It gives easy to control and clarity for the customer due to a strict reporting system.

Disadvantages of Waterfall model

- In this model, the risk factor is higher, so this model is not suitable for more significant and complex projects.
- This model cannot accept the changes in requirements during development.

INCREMENTAL MODEL

Incremental Model is a process of software development where requirements divided into multiple standalone modules of the software development cycle. In this model, each module goes through the requirements, design, implementation and testing phases. The process continues until the complete system achieved.

The various phases of incremental model are as follows

1. Requirement analysis: In the first phase of the incremental model, the product analysis expertise identifies the requirements. And the system functional requirements are understood by the requirement analysis team.

To develop the software under the incremental model, this phase performs a crucial role.

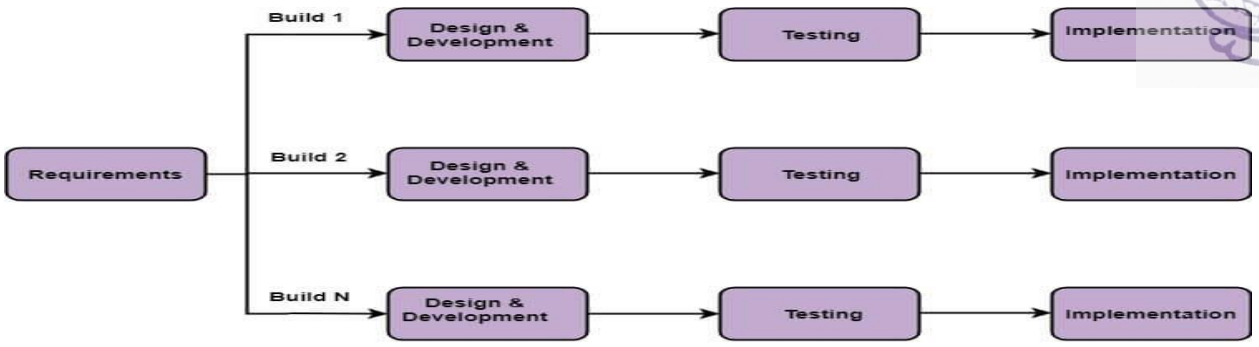


Fig: Incremental Model

2. Design & Development: In this phase of the Incremental model of SDLC, the design of the system functionality and the development method are finished with success. When software develops new practicality, the incremental model uses style and development phase.

3. Testing: In the incremental model, the testing phase checks the performance of each existing function as well as additional functionality. In the testing phase, the various methods are used to test the behavior of each task.

4. Implementation: Implementation phase enables the coding phase of the development system. It involves the final coding that design in the designing and development phase and tests the functionality in the testing phase. After completion of this phase, the number of the product working is enhanced and upgraded up to the final system product

Advantage of Incremental Model

- Errors are easy to be recognized.
- Easier to test and debug

- More flexible.
- Simple to manage risk

Disadvantage of Incremental Model

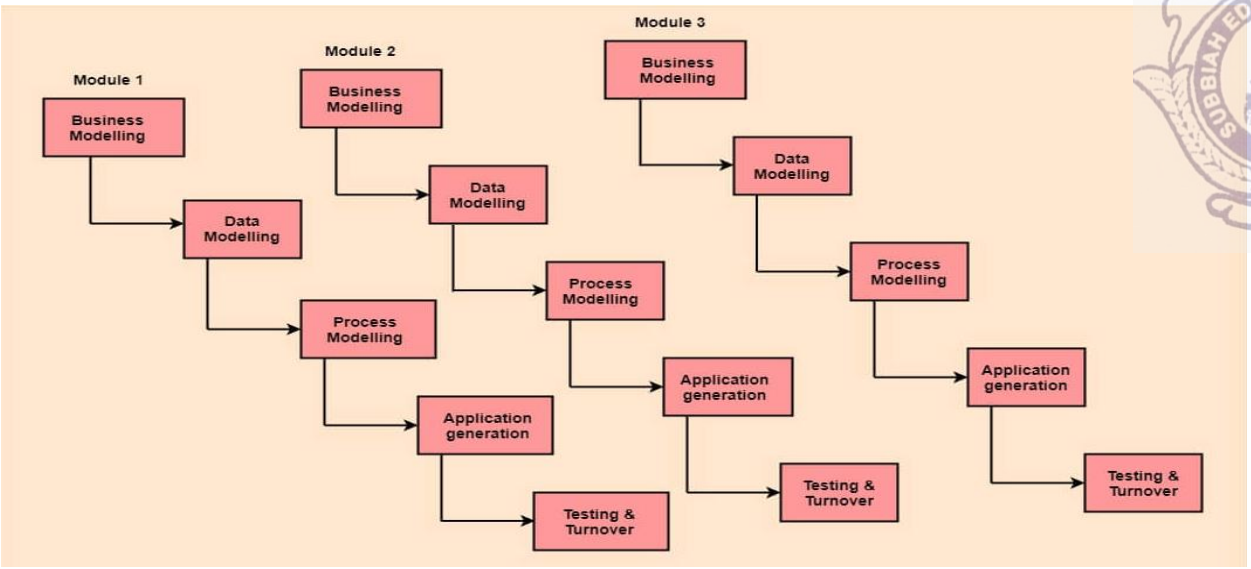
- Need for good planning
- Total Cost is high.

RAD (Rapid Application Development) Model

RAD is a linear sequential software development process model that emphasizes a concise development cycle using an element based construction approach. If the requirements are well understood and described, and the project scope is a constraint, the RAD process enables a development team to create a fully functional system within a concise time period.

RAD (Rapid Application Development) is a concept that products can be developed faster and of higher quality through:

- Gathering requirements using workshops or focus groups
- Prototyping and early, reiterative user testing of designs
- The re-use of software components
- A rigidly paced schedule that refers design improvements to the next product version
- Less formality in reviews and other team communication



The various phases of RAD are as follows

1.Business Modelling: The information flow among business functions is defined by answering questions like what data drives the business process, what data is generated, who generates it, where does the information go, who process it and so on.

2. Data Modelling: The data collected from business modeling is refined into a set of data objects (entities) that are needed to support the business. The attributes (character of each entity) are identified, and the relation between these data objects (entities) is defined.

3. Process Modelling: The information object defined in the data modeling phase are transformed to achieve the data flow necessary to implement a business function. Processing descriptions are created for adding, modifying, deleting, or retrieving a data object.

4. Application Generation: Automated tools are used to facilitate construction of the software; even they use the 4th GL techniques.

5. Testing & Turnover: Many of the programming components have already been tested since RAD emphasis reuse. This reduces the overall testing time. But the new part must be tested, and all interfaces must be fully exercised.

Advantage of RAD Model

- This model is flexible for change.
- In this model, changes are adoptable.
- It reduced development time.
- It increases the reusability of features.

Disadvantage of RAD Model

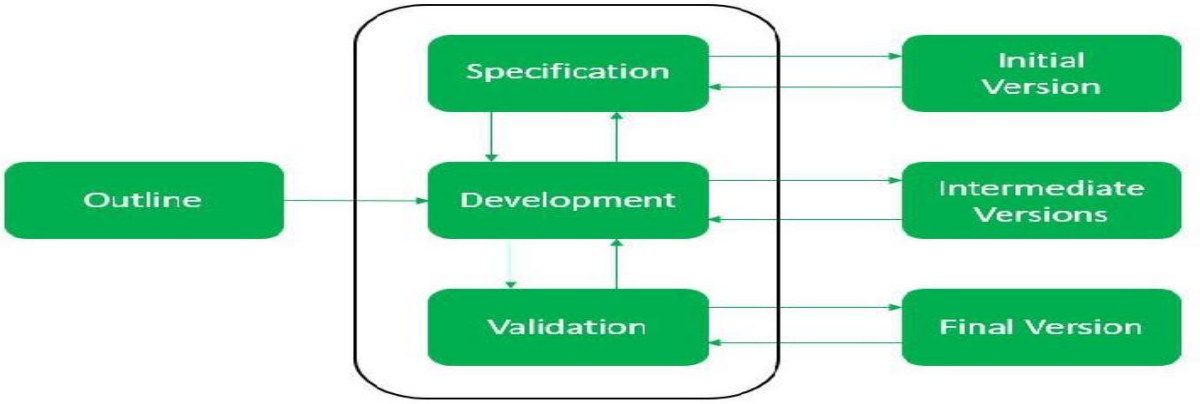
- It required highly skilled designers.
- All application is not compatible with RAD.
- For smaller projects, we cannot use the RAD model.
- On the high technical risk, it's not suitable.
- Required user involvement

EVOLUTIONARY PROCESS MODEL

The evolutionary model is based on the concept of making an initial product and then evolving the software product over time with iterative and incremental approaches with proper feedback.

In this type of model, the product will go through several iterations and come up when the final product is built through multiple iterations. The development is carried out simultaneously with the feedback during the development.

This model has a number of advantages such as customer involvement, taking feedback from the customer during development, and building the exact product that the user wants. Because of the multiple iterations, the chances of errors get reduced and the reliability and efficiency will increase.



Evolutionary Model

Advantages of the Evolutionary Process Model

1. During the development phase, the customer gives feedback regularly because the customer's requirement gets clearly specified.
2. After every iteration risk gets analyzed.
3. Suitable for big complex projects.
4. The first build gets delivered quickly as it used an iterative and incremental approach.

Disadvantages

- It is not suitable for small projects.

- The complexity of the spiral model can be more than the other sequential models.
- The cost of developing a product through a spiral model is high.

SPIRAL MODEL

The spiral model, initially proposed by Boehm, is an evolutionary software process model that couples the iterative feature of prototyping with the controlled and systematic aspects of the linear sequential model. Using the spiral model, the software is developed in a series of incremental releases.

The Spiral Model is shown in fig

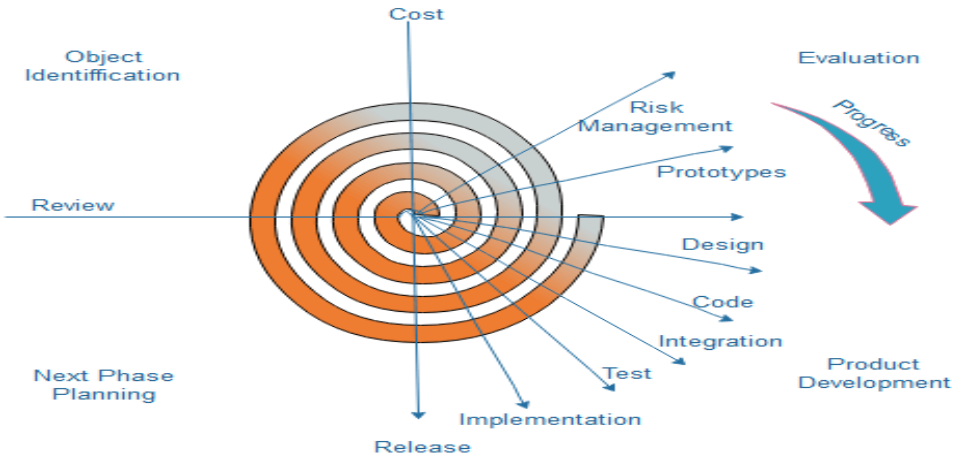


Fig. Spiral Model

Each cycle in the spiral is divided into four parts:

Objective setting: Each cycle in the spiral starts with the identification of purpose for that cycle, the various alternatives that are possible for achieving the targets, and the constraints that exists.

Risk Assessment and reduction: The next phase in the cycle is to calculate these various alternatives based on the goals and constraints.

The focus of evaluation in this stage is located on the risk perception for the project.

Development and validation: The next phase is to develop strategies that resolve uncertainties and risks. This process may include activities such as benchmarking, simulation, and prototyping.

Planning: Finally, the next step is planned. The project is reviewed, and a choice made whether to continue with a further period of the spiral. If it is determined to keep, plans are drawn up for the next step of the project.

The development phase depends on the remaining risks. For example, if performance or user-interface risks are treated more essential than the program development risks, the next phase may be an evolutionary development that includes developing a more detailed prototype for solving the risks.

The **risk-driven** feature of the spiral model allows it to accommodate any mixture of a specification-oriented, prototype-oriented, simulation-oriented, or another type of approach. An essential element of the model is that each period of the spiral is completed by a review that includes all the products developed during that cycle, including plans for the next cycle. The spiral model works for development as well as enhancement projects.

Advantages

- High amount of risk analysis
- Useful for large and mission-critical projects.

Disadvantages

- Can be a costly model to use.
- Risk analysis needed highly particular expertise
- Doesn't work well for smaller projects.

PROTOTYPE MODEL

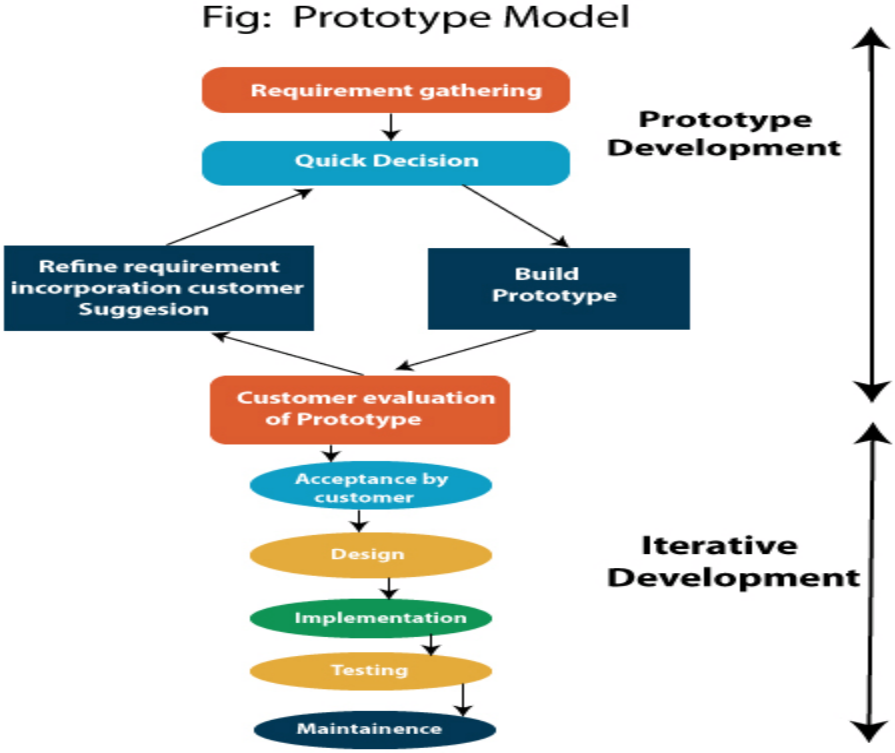
The prototype model requires that before carrying out the development of actual software, a working prototype of the system should be built. A prototype is a toy implementation of the system. A prototype usually turns out to be a very crude version of the actual system, possible exhibiting limited functional capabilities, low reliability, and inefficient performance as compared to actual software.

In many instances, the client only has a general view of what is expected from the software product. In such a scenario where there is an absence of detailed information regarding the input to the system, the processing needs, and the output requirement, the prototyping model may be employed.

Steps of Prototype Model

1. Requirement Gathering and Analyst
2. Quick Decision
3. Build a Prototype
4. Assessment or User Evaluation
5. Prototype Refinement
6. Engineer Product

Fig: Prototype Model



Advantage of Prototype Model

1. Reduce the risk of incorrect user requirement
2. Good where requirement are changing/uncommitted
3. Regular visible process aids management
4. Support early product marketing
5. Reduce Maintenance cost.
6. Errors can be detected much earlier as the system is made side by side.

Disadvantage of Prototype Model

1. An unstable/badly implemented prototype often becomes the final product.
2. Require extensive customer collaboration
    - Costs customer money
    - Needs committed customer

- Difficult to finish if customer withdraw
- May be too customer specific, no broad market

THE CONCURRENT DEVELOPMENT MODEL

• The concurrent development model is called as concurrent model.

• The communication activity has completed in the first iteration and exits in the awaiting changes state

. • The modeling activity completed its initial communication and then go to the underdevelopment state.

• If the customer specifies the change in the requirement, then the modeling activity moves from the under development state into the awaiting change state.

• The concurrent process model activities moving from one state to another state

Advantages of the concurrent development model

• This model is applicable to all types of software development processes.

• It is easy for understanding and use.

• It gives immediate feedback from testing.

• It provides an accurate picture of the current state of a project.

Disadvantages

• It needs better communication between the team members. This may not be achieved all the time.

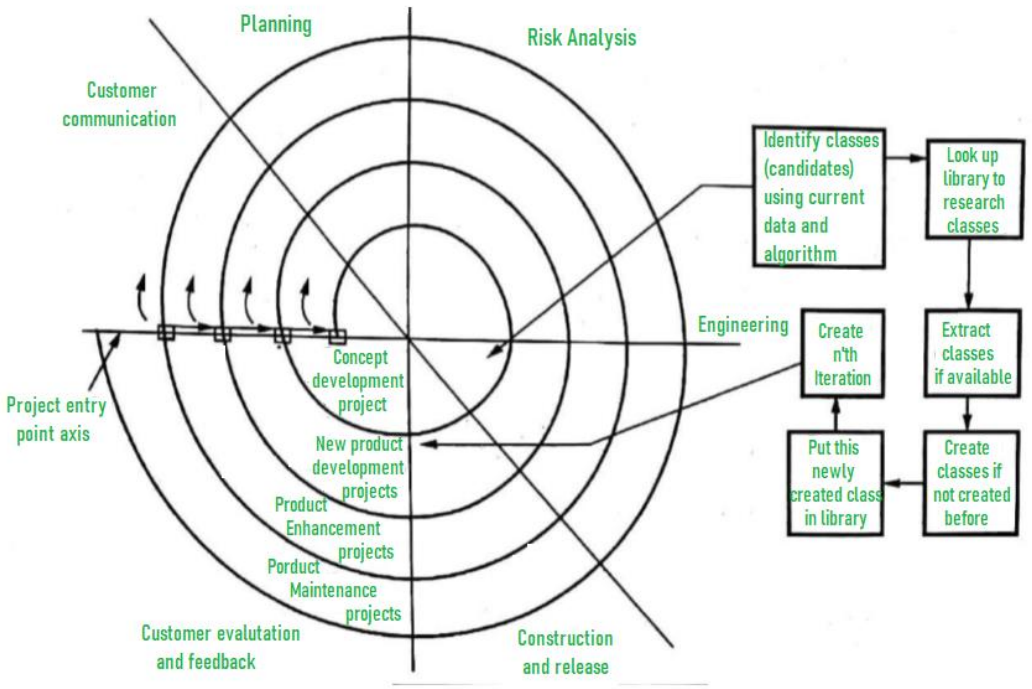
• It requires remembering the status of the different activities.

Component Based Model (CBM)

The component-based assembly model uses object-oriented technologies. In object-oriented technologies, the emphasis is on the creation of classes. Classes are the entities that encapsulate data and algorithms. In component-based architecture, classes (i.e., components required to build application) can be uses as reusable components.

This model uses various characteristics of spiral model. This model is evolutionary by nature. Hence, software development can be done using iterative approach. In CBD model, multiple classes can be used. These classes are basically the prepackaged components. The model works in following manner:

- **Step-1:** First identify all the required candidate components, i.e., classes with the help of application data and algorithms.
- **Step-2:** If these candidate components are used in previous software projects then they must be present in the library.
- **Step-3:** Such preexisting components can be excited from the library and used for further development.
- **Step-4:** But if the required component is not present in the library then build or create the component as per requirement.
- **Step-5:** Place this newly created component in the library. This makes one iteration of the system.
- **Step-6:** Repeat steps 1 to 5 for creating n iterations, where n denotes the number of iterations required to develop the complete application.



Characteristics of Component Assembly Model:

- Uses object-oriented technology.
- Components and classes encapsulate both data and algorithms.
- Components are developed to be reusable.
- Paradigm similar to spiral model, but engineering activity involves components.
- The system produced by assembling the correct components.

FORMAL METHODS

Formal methods are techniques we use in computer science and software engineering to ensure the correctness of our programs and reduction in our programs' errors. They rely on math and logic to model and analyze system behavior, making systems more reliable and secure.

Why do we use formal methods?

There are multiple reasons we use formal methods for:

- **Correctness and reliability:** Formal methods help solve errors and inconsistencies in software and hardware, making projects less error-prone, more reliable, and correct.
- **Early error detection:** Formal methods allow us to detect errors and inconsistencies in the early phases of our software development cycle, reducing future losses regarding time and money.
- **Fewer unambiguity:** By using formal languages for system specifications, we can clearly understand our system's behavior and specification, so there are fewer misunderstandings between us and the stakeholders.
- **Safety critical systems**: Safety critical systems refer to systems whose failure can lead to consequences of bigger impact, such as medical devices and aerospace, etc. Formal methods help us maintain high safety standards for such systems.
- **Verification and alidation:** We can use formal verification and validation methods to prove and ensure the correctness of our system's hardware and software.
- **Security:** Formal methods help us ensure the security of our devices by validating and verifying our systems' requirements.

Let's discuss the role of formal methods in phases of software development.

Analysis

Formal methods are useful to remove ambiguities and inconsistencies in our software application's requirements as it verifies and validates them

using formal verification and validation techniques. These techniques help us better understand the requirements.

Feasibility study

We use formal methods in feasibility studies of our software project to analyze its viability, assess risks, and ensure technical feasibility. This enhances accuracy in decision-making and reduction in potential errors that might occur in the future.

Design

Formal methods can be useful in the design phase of our software application as they can help in the software architecture using formal models to give us a road map for our software application.

Development

We can use formal specification methods in the development phase to remove any bugs and errors in implementing our code.

Testing

We can use formal verification methods for software components or the entire system to validate if the intent of the software application is fulfilled or not. Moreover, the software is tested using rigorous techniques to ensure a bug-free application.

Deployment

Formal methods can help in validating that our software application meets the client's requirements and needs and can ensure that the deployed software matches the formally verified models.

Maintenance

As changes are made to the software, formal methods can be used to verify that modifications do not introduce new issues or affect existing functionality.

Limitations of formal methods

- **Complexity:** For larger and more complex systems, formal methods are difficult to apply. It can be more time and effort-consuming.
- **Expensive:** Formal methods are costly in terms of time and resources as compared to traditional verification and validation techniques.
- **Limited Scope:** Formal methods are suitable for a limited type of system. For systems with rapidly changing requirements or a high degree of uncertainty, formal methods is not suitable.
- **Difficult to understand:** Formal methods are highly precise and detailed, so they can be difficult to understand for some complex systems.

Aspect-Oriented Software Development

Aspect-oriented software development (AOSD) is a software design solution that helps address the modularity issues that are not properly resolved by other software approaches, like procedural, structured and object-oriented programming (OOP). AOSD complements, rather than replaces, these other types of software approaches.AOSD is also known as aspect-oriented programming (AOP).

Aspect-Oriented Software Development

AOSD features are as follows:

- Considered a subset of post-object programming technologies
- Better software design support through isolating application business logic from supporting and secondary functions
- Provides complementary benefits and may be used with other agile processes and coding standards
- Key focus – Identification, representation and specification of concerns, which also may be cross-cutting
- Provides better modularization support of software designs, reducing software design, development and maintenance costs
- Modularization principle based on involved functionalities and processes
- Because concerns are encapsulated into different modules, localization of crosscutting concerns is better promoted and handled
- Provides tools and software coding techniques to ensure modular content support at the source code level
- Promotes reusability of code used for the modularization of cross-cutting concerns
- Smaller code size, due to tackling cross cutting concerns
- REDUCED EFFICIENCY FROM INCREASED OVERHEAD

AGILE MODEL/AGILITY

The agile manifesto also called manifesto for agile software development.The meaning of Agile is swift or versatile."**Agile process model**" refers to a software development approach based on iterative development.

Agile methods break tasks into smaller iterations, or parts do not directly involve long term planning. The project scope and requirements are laid down at the beginning of the development process. Plans regarding the number of iterations, the duration and the scope of each iteration are clearly defined in advance.

Each iteration is considered as a short time "frame" in the Agile process model, which typically lasts from one to four weeks. The division of the entire project into smaller parts helps to minimize the project risk and to reduce the overall project delivery time requirements. Each iteration involves a team working through a full software development life cycle including planning, requirements analysis, design, coding, and testing

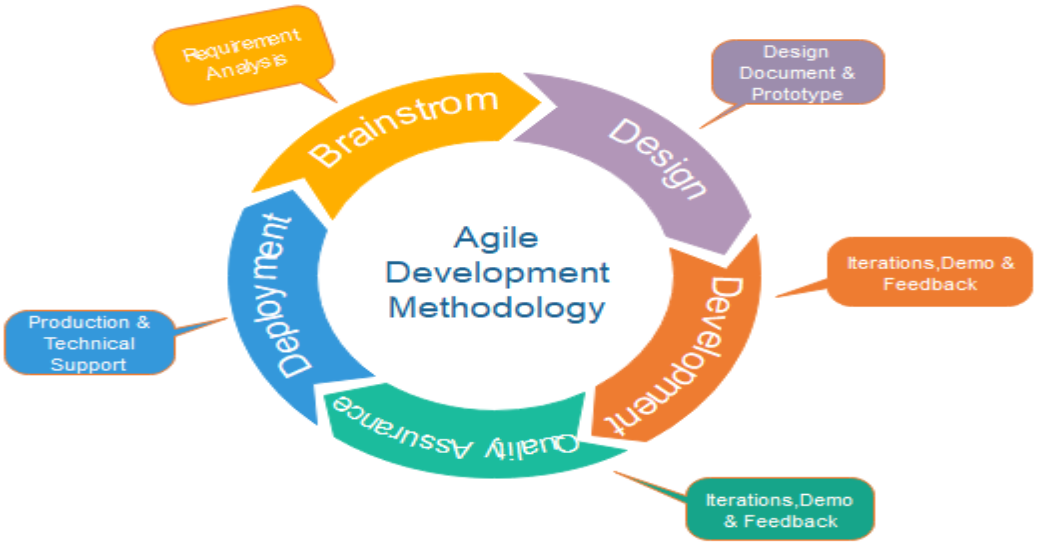


Fig. Agile Model

Phases of Agile Model

Following are the phases in the Agile model are as follows:

1. Requirements gathering
2. Design the requirements
3. Construction/ iteration

4. Testing/ Quality assurance
5. Deployment
6. Feedback

1. Requirements gathering: In this phase, you must define the requirements. You should explain business opportunities and plan the time and effort needed to build the project. Based on this information, you can evaluate technical and economic feasibility.

2. Design the requirements: When you have identified the project, work with stakeholders to define requirements. You can use the user flow diagram or the high-level UML diagram to show the work of new features and show how it will apply to your existing system.

3. Construction/ iteration: When the team defines the requirements, the work begins. Designers and developers start working on their project, which aims to deploy a working product. The product will undergo various stages of improvement, so it includes simple, minimal functionality.

4. Testing: In this phase, the Quality Assurance team examines the product's performance and looks for the bug.

5. Deployment: In this phase, the team issues a product for the user's work environment.

6. Feedback: After releasing the product, the last step is feedback. In this, the team receives feedback about the product and works through the feedback.

The following 12 Principles are based on the Agile Manifesto.

Our highest priority is to satisfy the customer through the early and continuous delivery of valuable software.

Working software is the primary measure of progress.

Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.

Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

Continuous attention to technical excellence and good design enhances agility.

Business people and developers must work together daily throughout the project.

Simplicity–the art of maximizing the amount of work not done–is essential.

Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

The best architectures, requirements, and designs emerge from self-organizing teams.

The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Agile Testing Methods:

- Scrum
- Crystal
- Dynamic Software Development Method(DSDM)
- Feature Driven Development(FDD)

- Lean Software Development
- eXtreme Programming(XP)

Scrum

SCRUM is an agile development process focused primarily on ways to manage tasks in team-based development conditions.

There are three roles in it, and their responsibilities are:

- **Scrum Master:** The scrum can set up the master team, arrange the meeting and remove obstacles for the process
- **Product owner:** The product owner makes the product backlog, prioritizes the delay and is responsible for the distribution of functionality on each repetition.
- **Scrum Team:** The team manages its work and organizes the work to complete the sprint or cycle.

eXtreme Programming(XP)

This type of methodology is used when customers are constantly changing demands or requirements, or when they are not sure about the system's performance.

Dynamic Software Development Method (DSDM):

DSDM is a rapid application development strategy for software development and gives an agile project distribution structure. The techniques used in DSDM are:

1. Time Boxing
2. MoSCoW Rules

3. Prototyping

The DSDM project contains seven stages:

1. Pre-project
2. Feasibility Study
3. Business Study
4. Functional Model Iteration
5. Design and build Iteration
6. Implementation
7. Post-project

Feature Driven Development(FDD):

This method focuses on "Designing and Building" features. In contrast to other smart methods, FDD describes the small steps of the work that should be obtained separately per function.

Lean Software Development:

Lean software development methodology follows the principle "just in time production." The lean method indicates the increasing speed of software development and reducing costs.

Advantage(Pros) of Agile Method:

1. Frequent Delivery
2. Face-to-Face Communication with clients.
3. Efficient design and fulfils the business requirement.
4. Anytime changes are acceptable.
5. It reduces total development time.

Disadvantages(Cons) of Agile Model:

1. Due to the shortage of formal documents, it creates confusion and crucial decisions taken throughout various phases can be misinterpreted at any time by different team members.
2. Due to the lack of proper documentation, once the project completes and the developers allotted to another project, maintenance of the finished project can become a difficulty.

What is Extreme Programming?

XP is a lightweight, efficient, low-risk, flexible, predictable, scientific, and fun way to develop a software.e**X**treme **P**rogramming (XP) was conceived and developed to address the specific needs of software development by small teams in the face of vague and changing requirements.

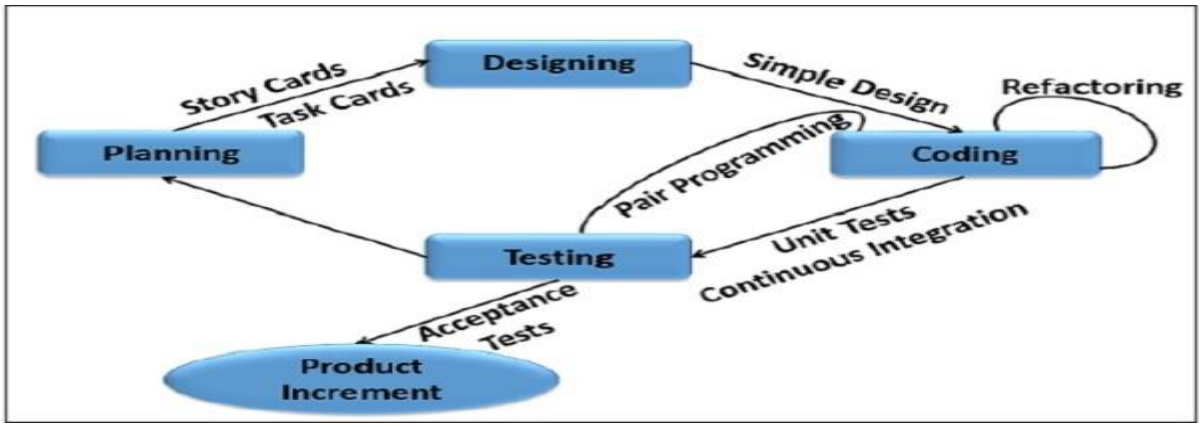
Extreme Programming builds on these activities and coding. It is the detailed (not the only) design activity with multiple tight feedback loops through effective implementation, testing and refactoring continuously.

Extreme Programming is based on the following values −

- Communication
- Simplicity
- Feedback
- Courage
- Respect

Extreme Programming is one of the Agile software development methodologies. It provides values and principles to guide the team behavior. The team is expected to self-organize. Extreme Programming provides specific core practices where −

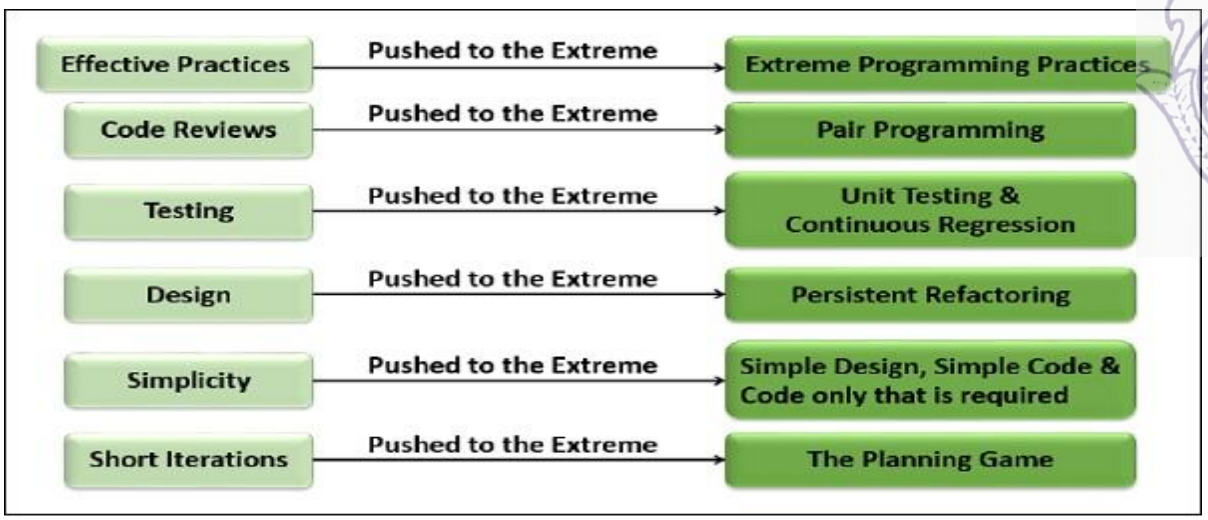
- Each practice is simple and self-complete.
- Combination of practices produces more complex and emergent behavior.



Why is it called “Extreme?”

Extreme Programming takes the effective principles and practices to extreme levels.

- Code reviews are effective as the code is reviewed all the time.
- Testing is effective as there is continuous regression and testing.
- Design is effective as everybody needs to do refactoring daily.
- Integration testing is important as integrate and test several times a day.
- Short iterations are effective as the planning game for release planning and iteration planning.



History of Extreme Programming

Kent Beck, Ward Cunningham and Ron Jeffries formulated extreme Programming in 1999. The other contributors are Robert Martin and Martin Fowler.

Extreme Programming Advantages

Extreme Programming solves the following problems often faced in the software development projects −

- **Slipped schedules** − and achievable development cycles ensure timely deliveries.
- **Cancelled projects** − Focus on continuous customer involvement ensures transparency with the customer and immediate resolution of any issues.
- **Costs incurred in changes** − Extensive and ongoing testing makes sure the changes do not break the existing functionality. A running working system always ensures sufficient time for accommodating changes such that the current operations are not affected.

- **Production and post-delivery defects: Emphasis is on** − the unit tests to detect and fix the defects early.
- **Misunderstanding the business and/or domain** − Making the customer a part of the team ensures constant communication and clarifications.
- **Business changes** − Changes are considered to be inevitable and are accommodated at any point of time.
- **Staff turnover** − Intensive team collaboration ensures enthusiasm and good will. Cohesion of multi-disciplines fosters the team spirit.

Good practices need to be practiced in extreme programming:

Some of the good practices that have been recognized in the extreme programming model and suggested to maximize their use are given below:

- **Code Review:** Code review detects and corrects errors efficiently. It suggests pair programming as coding and reviewing of written code carried out by a pair of programmers who switch their work between them every hour.

- **Testing:** Testing code helps to remove errors and improves its reliability. XP suggests test-driven development (TDD) to continually write and execute test cases. In the TDD approach, test cases are written even before any code is written.

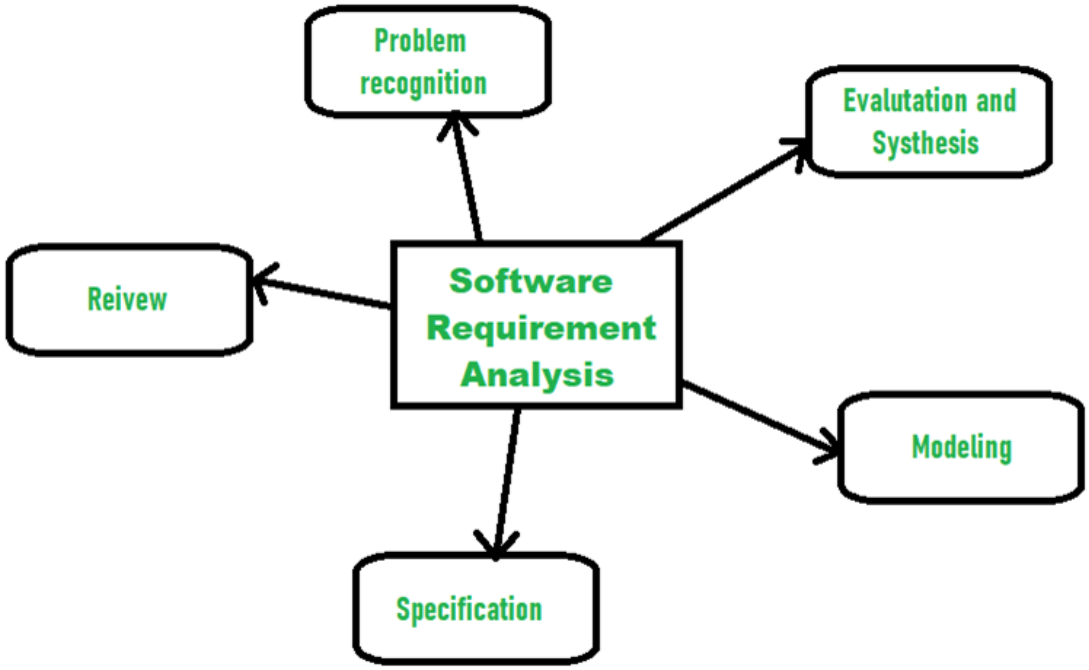
- **Incremental development:** Incremental development is very good because customer feedback is gained and based on this development team comes up with new increments every few days after each iteration.

- **Simplicity:** Simplicity makes it easier to develop good-quality code as well as to test and debug it.
- **Design:** Good quality design is important to develop good quality software. So, everybody should design daily.
- **Integration testing:** It helps to identify bugs at the interfaces of different functionalities. Extreme programming suggests that the developers should achieve continuous integration by building and performing integration testing several times a day.

UNIT II

1.REQUIREMENTS ANALYSIS

Software requirement analysis simply means complete study, analyzing, describing software requirements so that requirements that are genuine and needed can be fulfilled to solve problem. There are several activities involved in analyzing Software requirements. Some of them are given below



Requirements analysis and feature creep

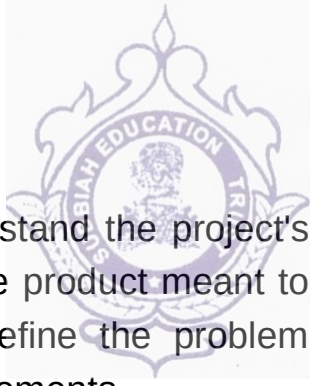
Requirements analysis and clear communication help to prevent feature creep in software projects. Also known as *scope creep*, feature creep refers to the addition of excessive features to a product, which often makes it overly complex and difficult to use.

Requirements analysis process

Requirements analysis is a multistage process that involves the following steps.

1. Understand the key stakeholders and end users

In any project, the key stakeholders, including end users, generally have final say on the project scope. Project teams should identify them early and involve them in the requirements gathering process from the beginning.



2. Understand the project goal

To capture all necessary requirements, project teams must first understand the project's objective. What business need drives the project? What problem is the product meant to solve? By understanding the desired end, the project team can define the problem statement and have more productive discussions when gathering requirements.

3. Capture requirements

At this stage, all relevant stakeholders provide requirements. This can be done through one-on-one interviews, focus groups or consideration of use cases. Project teams gather stakeholder feedback and incorporate it into requirements.

4. Categorize requirements

Categorization of requirements can help with prioritization, impact analysis, feasibility analysis and conflict resolution. Four common requirements categories are the following:

1. Functional requirements.
2. Technical requirements.
3. Transitional requirements.
4. Operational requirements.

5. Interpret and document requirements

Post-categorization, the project team should analyze its set of requirements to determine which ones are feasible. Interpretation and analysis are easier when requirements are well defined and clearly worded. Each requirement should have a clear and understood impact on the end product and the project plan. After all the requirements have been identified, prioritized and analyzed, project teams should document them in the software requirements specification (SRS).

6. Finalize SRS and get sign-off on requirements

The SRS should be shared with key stakeholders for sign-off to ensure that they agree with the requirements. This helps prevent conflicts or disagreements later. Feedback, if any, should be incorporated. The SRS can then be finalized and made available to the entire development team. This document provides the foundation for the project's scope and guides other steps during the software development lifecycle (SDLC), including development and testing.

2. REQUIREMENT GATHERINGR

A requirement gathering is the process of identifying your project's exact requirements from start to finish. This process occurs during the project initiation phase but you'll continue to manage your project requirements throughout the entire project timeline.

Step 1: Assign roles

The first step in requirements gathering is to assign roles in your project. This is when you identify your project stakeholders.

A stakeholder is anyone invested in the project, whether they're internal or external partners. For example, a customer is an external stakeholder, while a department manager or board member is an internal stakeholder. Identifying these roles first will help you determine who should analyze your project scope later on.

Other roles include the project manager, project administrator, designers, product testers, and developers. These people can help you identify the requirements and resources you need in order to hit your project goals.

While you may feel tempted to jump headfirst into your project and start listing all the things you know you'll need, this can be a mistake. Slow down and stick to the process and you'll have a better chance of preventing project risk.

Step 2: Meet with stakeholders

Once you've identified your project stakeholders, meet with them to get an idea of what they're hoping to get out of the project. Understanding what stakeholders want matters because they're ultimately the ones you're creating your deliverables for.

Some questions you can ask include:

- What is your goal for this project?
- What do you think would make this project successful?
- What are your concerns about this project?
- What do you wish this product or service would do that it doesn't already?
- What changes would you recommend about this project?

The stakeholders are the people you're ultimately developing the project for, so you should ask them questions that can help you create your list of requirements.

Step 3: Gather and document

Step three in the process happens at the same time as step two. You'll gather information as you ask your stakeholders questions. The goal is to document everything you can, so have all of the answers you need to start your project.

Use a project management tool to collect and document this information. That way, you can keep your project plan, project requirements, and project communication all in one place. Some examples of what you might document include:

- Stakeholder answers to interview questions
- Stakeholder questions
- Stakeholder requests
- Stakeholder comments
- Questions and comments that arise during interviews

You don't have to use every answer you receive, but having everything documented can help you see all of your stakeholders' perspectives, which will help you with requirements management.

Step 4: List assumptions and requirements

Now that you've completed the intake process, create your requirements management plan based on the information you've gathered.

Consider the questions you initially set out to answer during the requirements gathering process. Then, use them to create your requirements goals, including:

- Length of project schedule: You can map out your project timeline using a Gantt chart and use it to visualize any project requirements that depend on project milestones. Some requirements will apply for the full duration of the project, whereas others may only apply during distinct project phases. For example, you'll need a specific budget for team member salaries throughout the entire project, but you may only need specific material during the last stage of your project timeline.
- People involved in the project: Identify exactly which team members will be involved in your project, including how many designers, developers, or managers you'll need to execute every step. People are part of your project requirements because if you don't have the team members you need, you won't be able to complete the project on time.
- Project risks: Understanding your project risks is an important part of identifying project requirements. Use a risk register to determine which risks are of highest priority, such as stakeholder feedback, timeline delays, and lack of budget. Then, schedule a brainstorming session with your team to figure out how to prevent these risks.

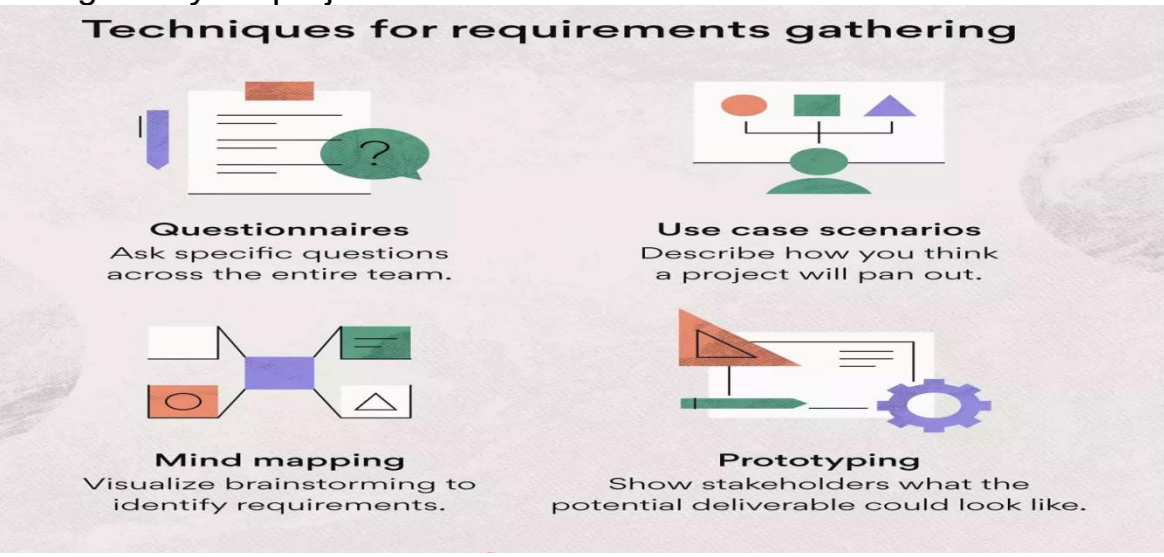
Like SMART goals, your project requirements should be actionable, measurable, and quantifiable. Try to go into as much detail as possible when listing out your project budget, timeline, required resources, and team.

Step 5: Get approval

Once you formalize your project requirements, you'll need approval from stakeholders to ensure you're meeting user needs. Encouraging clear communication can also prevent scope creep by ensuring your stakeholders know the limits of the project from the beginning. You can then proceed with your implementation plan, which may include acquiring resources and assembling a team.

Step 6: Monitor progress

The last part of the process is monitoring the progress of your project. You can use project management software to track your project budget and other requirements as you move through project execution. The benefit of project management software is that you can see changes to your project in real-time and take immediate action when things go awry.



3.FUNCTIONAL VS NON FUNCTIONAL REQUIREMENTS

Functional Requirements

These are the requirements that the end user specifically demands as basic facilities that the system should offer. All these functionalities need to be necessarily incorporated into the system as a part of the contract.

These are represented or stated in the form of input to be given to the system, the operation performed and the output expected. They are the requirements stated by the user which one can see directly in the final product, unlike the non-functional requirements.

Example:

- What are the features that we need to design for this system?
- What are the edge cases we need to consider, if any, in our design?

Non-Functional Requirements

These are the quality constraints that the system must satisfy according to the project contract. The priority or extent to which these factors are implemented varies from one project to another. They are also called non-behavioral requirements. They deal with issues like:

- Portability
- Security
- Maintainability

- Reliability
- Scalability
- Performance
- Reusability
- Flexibility

Example:

- Each request should be processed with the minimum latency?
- System should be highly valuable.

Extended Requirements

These are basically “nice to have” requirements that might be out of the scope of the System.

Example:

- Our system should record metrices and analytics.
- Service heath and performance monitoring.

Difference between Functional Requirements and Non-Functional Requirements

Functional Requirements	Non Functional Requirements
A functional requirement defines a system or its component.	A non-functional requirement defines the quality attribute of a software system.
It specifies “What should the software system do?”	It places constraints on “How should the software system fulfill the functional requirements?”
Functional requirement is specified by User.	Non-functional requirement is specified by technical peoples e.g. Architect, Technical leaders and software developers.
It is mandatory.	It is not mandatory.
It is captured in use case.	It is captured as a quality attribute.
Defined at a component level.	Applied to a system as a whole.

4.FEASIBILITY STUDY

Feasibility Study in Software Engineering is a study to evaluate feasibility of proposed project or system. Feasibility study is one of stage among important four stages of Software Project Management Process. As name suggests feasibility study is the feasibility analysis or it is a measure of the software product in terms of how much beneficial product development will be for the organization in a practical point of view. Feasibility study is carried out based on many purposes to analyze whether software product will be right in terms of development, implantation, contribution of project to the organization etc.

TYPES OF FEASIBILITY STUDY

The feasibility study mainly concentrates on below five mentioned areas. Among these Economic Feasibility Study is most important part of the feasibility analysis and Legal Feasibility Study is less considered feasibility analysis.

1. **Technical Feasibility**: This technical feasibility study gives report whether there exists correct required resources and technologies which will be used for project development. Along with this, feasibility study also analyzes technical skills and capabilities of technical team, existing technology can be used or not,
2. **Operational Feasibility:** In Operational Feasibility degree of providing service to requirements is analyzed along with how much easy product will be to operate and maintenance after deployment.

3. **Economic Feasibility:** In Economic Feasibility study cost and benefit of the project is analyzed.
4. **Legal Feasibility:** In Legal Feasibility study project is analyzed in legality point of view. Overall it can be said that Legal Feasibility if proposed project conform legal and ethical requirements.
5. **Schedule Feasibility:** In Schedule Feasibility Study mainly timelines/deadlines is analyzed for proposed project which includes how many times teams will take to complete final project which has a great impact on the organization as purpose of project may fail if it can't be completed on time.
7. **Cultural and Political Feasibility:** This section assesses how the software project will affect the political environment and organizational culture.
8. **Market Feasibility:** This refers to evaluating the market's willingness and ability to accept the suggested software system. Analyzing the target market, understanding consumer wants and assessing possible rivals are all part of this study.
9. **Resource Feasibility:** This method evaluates if the resources needed to complete the software project successfully It guarantees that sufficient hardware, software, trained labor and funding are available to complete the project successfully.

Aim of Feasibility Study

- The overall objective of the organization are covered and contributed by the system or not.
- The implementation of the system be done using current technology or not.

- Can the system be integrated with the other system which are already exist

Feasibility Study Process

The below steps are carried out during entire feasibility analysis.

1. Information assessment: It assesses the original project concept and establishes the main aims and objectives.
2. Information collection: It collects the necessary information and data required to evaluate the project's many components.
3. Report writing: It produces an in-depth feasibility report that details the analysis and results.
4. General information: It gives a summary of the main points discussed in the report on the feasibility study.

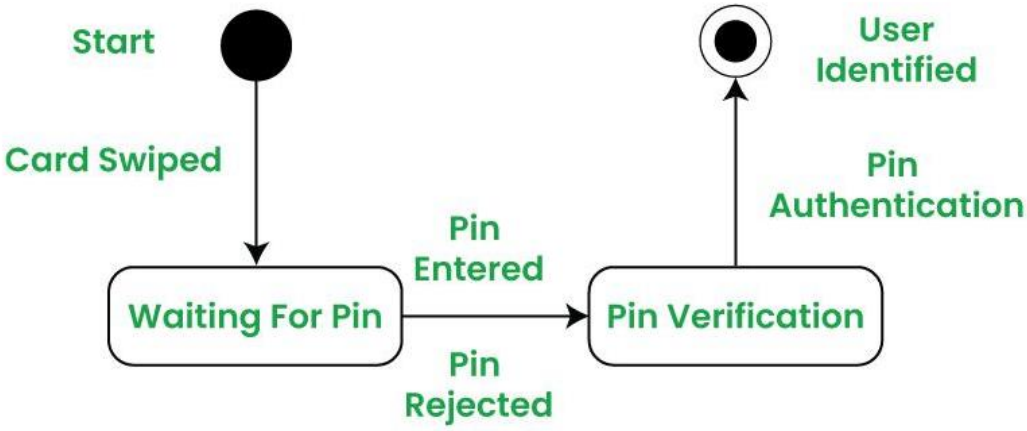
STATE MACHINE DIAGRAMS

- State Machine diagrams are also referred to as State Machines Diagrams and State-Chart Diagrams.
- These terms are often used interchangeably. So simply, a state machine diagram is used to model the dynamic behavior of a class in response to time and changing external stimuli.
- We can say that each and every class has a state but we don't model every class using State Machine diagrams.
- We prefer to model the states with three or more states.

Let's understand State Machine Diagram with the help of an example:

Example:

A State Machine Diagram for user verification



State Machine Diagrams | Unified Modeling Language (UML)

The State Machine Diagram above shows the different states in which the verification sub-system or class exist for a particular system.

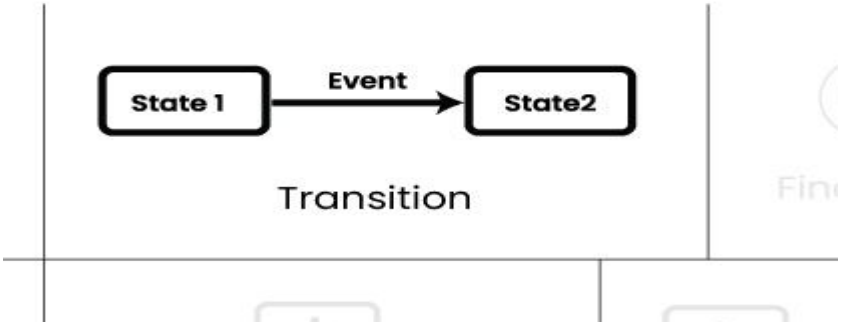
2. Basic components and notations of a State Machine diagram

2.1. Initial state

We use a black filled circle represent the initial state of a System or a Class.

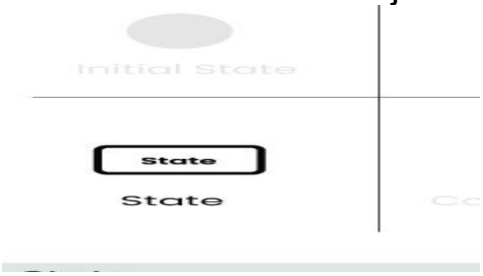
2.2. Transition

We use a solid arrow to represent the transition or change of control from one state to another. The arrow is labelled with the event which causes the change in state.



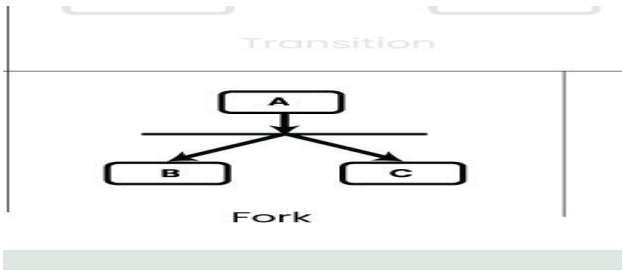
2.3. State

We use a rounded rectangle to represent a state. A state represents the conditions or circumstances of an object of a class at an instant of time.



2.4. Fork

We use a rounded solid rectangular bar to represent a Fork notation with incoming arrow from the parent state and outgoing arrows towards the newly created states. We use the fork notation to represent a state splitting into two or more concurrent states.

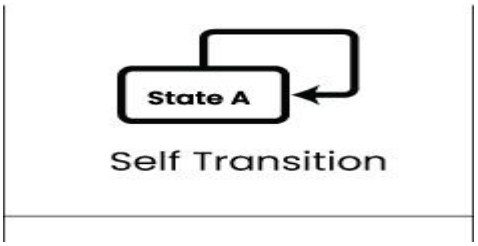


2.5. Join

We use a rounded solid rectangular bar to represent a Join notation with incoming arrows from the joining states and outgoing arrow towards the common goal state. We use the join notation when two or more states concurrently converge into one on the occurrence of an event or events.

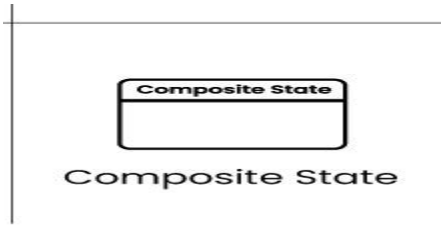
2.6. Self transition

We use a solid arrow pointing back to the state itself to represent a self transition. There might be scenarios when the state of the object does not change upon the occurrence of an event. We use self transitions to represent such cases.



2.7. Composite state

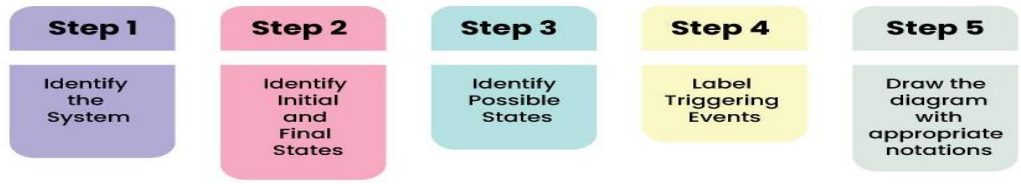
We use a rounded rectangle to represent a composite state also. We represent a state with internal activities using a composite state.



2.8. Final State

We use a filled circle within a circle notation to represent the final state in a state machine diagram.

3. How to draw a State Machine diagram in UML?



State Machine Diagrams | Unified Modeling Language (UML)

Below are the steps of how to draw the State Machine Diagram in UML:

Step1. Identify the System:

- Understand what your diagram is representing.
- Whether it's a machine, a process, or any object, know what different situations or conditions it might go through.

Step2. Identify Initial and Final States:

- Figure out where your system starts (initial state) and where it ends (final state).
- These are like the beginning and the end points of your system's journey.

Step3. Identify Possible States:

- Think about all the different situations your system can be in.
- These are like the various phases or conditions it experiences.
- Use boundary values to guide you in defining these states.

Step4. Label Triggering Events:

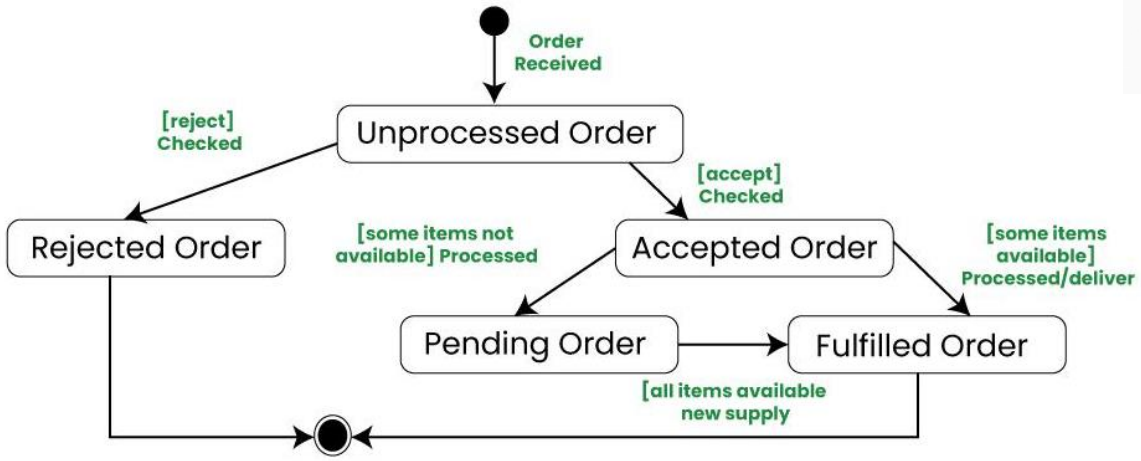
- Understand what causes your system to move from one state to another.
- These causes or conditions are the events.
- Label each transition with what makes it happen.

Step5. Draw the Diagram with appropriate notations:

- Now, take all this information and draw it out.
- Use rectangles for states, arrows for transitions, and circles or rounded rectangles for initial and final states.
- Be sure to connect everything in a way that makes sense.

Let’s understand State Machine diagram with the help of an example, ie for an online order :

State machine diagram for an online order



State Machine Diagrams | Unified Modeling Language (UML)

The UML diagrams we draw depend on the system we aim to represent. Here is just an example of how an online ordering system might look like :

1. On the event of an order being received, we transit from our initial state to Unprocessed order state.
2. The unprocessed order is then checked.
3. If the order is rejected, we transit to the Rejected Order state.
4. If the order is accepted and we have the items available we transit to the fulfilled order state.
5. However if the items are not available we transit to the Pending Order state.
6. After the order is fulfilled, we transit to the final state. In this example, we merge the two states i.e. Fulfilled order and Rejected order into one final state.

UNIT III

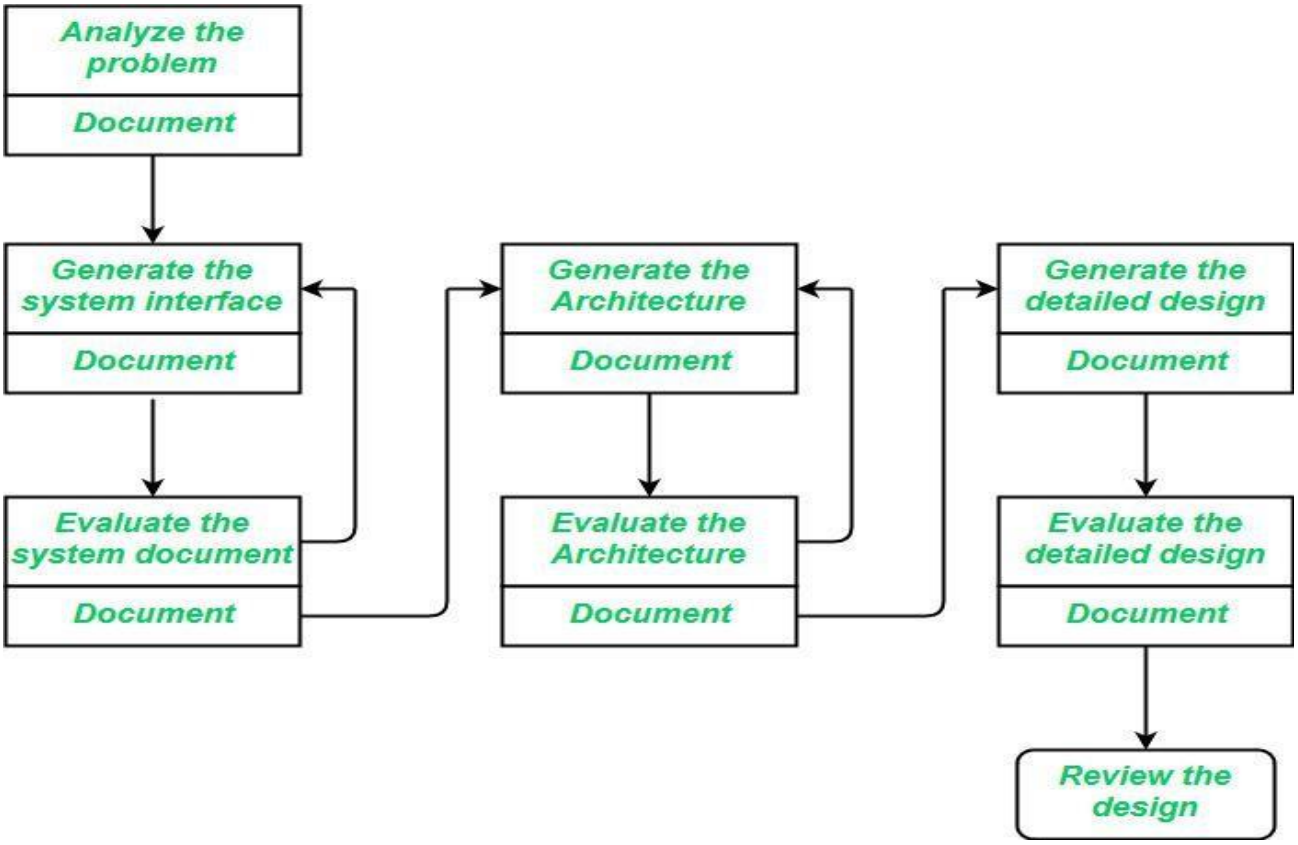
SOFTWARE DESIGN

Unit-3

Software Design Process

The design phase of software development deals with transforming the customer requirements as described in the SRS documents into a form implementable using a programming language.
The software design process can be divided into the following three levels of phases of design:

1. Interface Design
2. Architectural Design
3. Detailed Design



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 focussed 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 on 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.

The architectural design adds important details ignored during the interface design. Design of the internals of the major components is ignored until the last phase of the design.

Detailed Design:

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
- Algorithms and data structures

Introduction to design process

- The main aim of design engineering is to generate a model which shows firmness, delight and commoditySoftware design is an iterative process through which requirements are translated into the blueprint for building the software.

Software quality guidelines

- A design is generated using the recognizable architectural styles and compose a good design characteristic of components and it is implemented in evolutionary manner for testing.
- A design of the software must be modular i.e the software must be logically partitioned into elements.
- In design, the representation of data , architecture, interface and components should be distinct.
- A design must carry appropriate data structure and recognizable data patterns.

- Design components must show the independent functional characteristic.
- A design creates an interface that reduce the complexity of connections between the components.
- A design must be derived using the repeatable method.
- The notations should be use in design which can effectively communicates its meaning.

Quality attributes

The attributes of design name as 'FURPS' are as follows:

Functionality:

It evaluates the feature set and capabilities of the program.

Usability:

It is accessed by considering the factors such as human factor, overall aesthetics, consistency and documentation.

Reliability:

It is evaluated by measuring parameters like frequency and security of failure, output result accuracy, the mean-time-to-failure(MTTF), recovery from failure and the the program predictability.

Performance: It is measured by considering processing speed, response time, resource consumption, throughput and efficiency.

Supportability:

- It combines the ability to extend the program, adaptability, serviceability.

  These three term defines the maintainability.

- Testability, compatibility and configurability are the terms using which a system can be easily installed and found the problem easily.
- Supportability also consists of more attributes such as compatibility, extensibility, fault tolerance, modularity, reusability, robustness, security, portability, scalability.

Design concepts

The set of fundamental software design concepts are as follows:

1. Abstraction

- A solution is stated in large terms using the language of the problem environment at the highest level abstraction.
- The lower level of abstraction provides a more detail description of the solution.
- A sequence of instruction that contain a specific and limited function refers in a procedural abstraction.
- A collection of data that describes a data object is a data abstraction.

2. Architecture

- The complete structure of the software is known as software architecture.
- Structure provides conceptual integrity for a system in a number of ways.
- The architecture is the structure of program modules where they interact with each other in a specialized way.
- The components use the structure of data.
- The aim of the software design is to obtain an architectural framework of a system.
- The more detailed design activities are conducted from the framework.

3. Patterns

A design pattern describes a design structure and that structure solves a particular design problem in a specified content.

4. Modularity

- A software is separately divided into name and addressable components. Sometime they are called as modules which integrate to satisfy the problem requirements.

Modularity is the single attribute of a software that permits a program to be managed easily.

5. Information hiding

Modules must be specified and designed so that the information like algorithm and data presented in a module is not accessible for other modules not requiring that information.

6. Functional independence

- The functional independence is the concept of separation and related to the concept of modularity, abstraction and information hiding.
- The functional independence is accessed using two criteria i.e Cohesion and coupling.

Cohesion

- Cohesion is an extension of the information hiding concept.
- A cohesive module performs a single task and it requires a small interaction with the other components in other parts of the program.

Coupling

Coupling is an indication of interconnection between modules in a structure of software.

7. Refinement

- Refinement is a top-down design approach.
- It is a process of elaboration.

- A program is established for refining levels of procedural details.
- A hierarchy is established by decomposing a statement of function in a stepwise manner till the programming language statement are reached.

8. Refactoring

- It is a reorganization technique which simplifies the design of components without changing its function behaviour.
- Refactoring is the process of changing the software system in a way that it does not change the external behaviour of the code still improves its internal structure.

9. Design classes

Software Engineering | Architectural Design

Introduction: The software needs the architectural design to represents the design of 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.

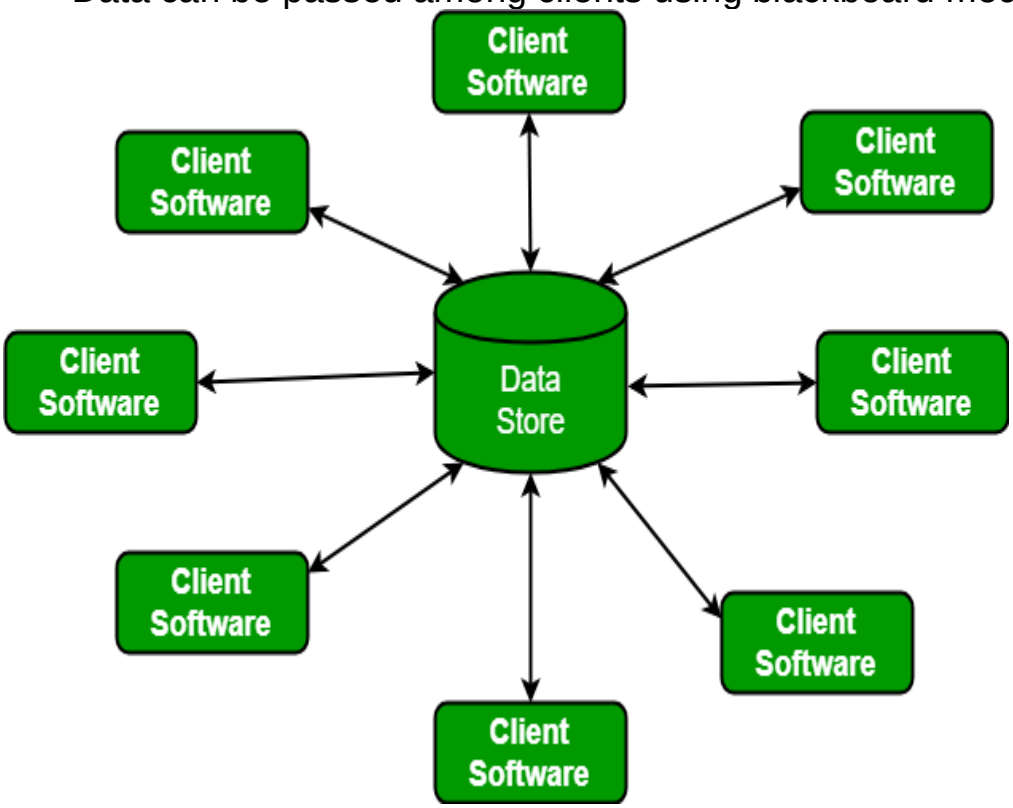
Each style will describe a system category that consists of :A set of components(eg: a database, computational modules) that will perform a function required by the system.

- The set of connectors will help in coordination, communication, and cooperation between the components.
- Conditions that how components can be integrated to form the system.
- Semantic models that help the designer to understand the overall properties of the system. The use of architectural styles is to establish a structure for all the components of the system. Taxonomy of Architectural styles:

1. **Data centred 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

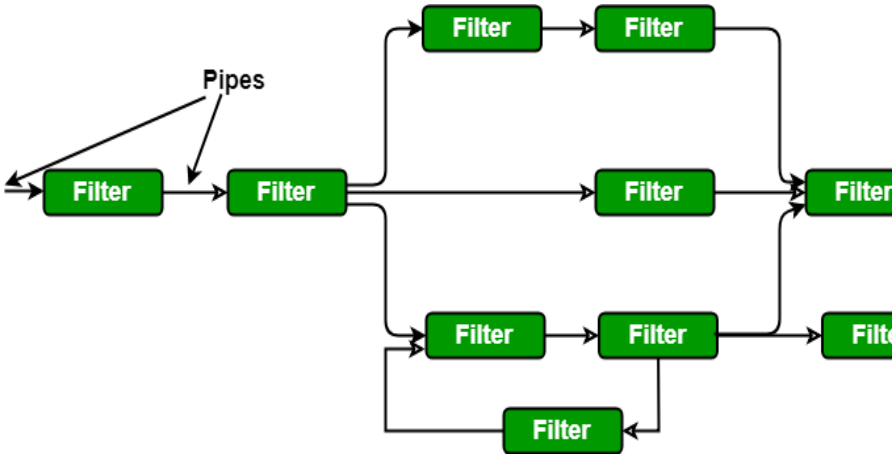
software access a central repository. Variation of this approach are used to transform the repository into a blackboard when data related to client or data of interest for the client change the notifications to client software.

- 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 blackboard mechanism.



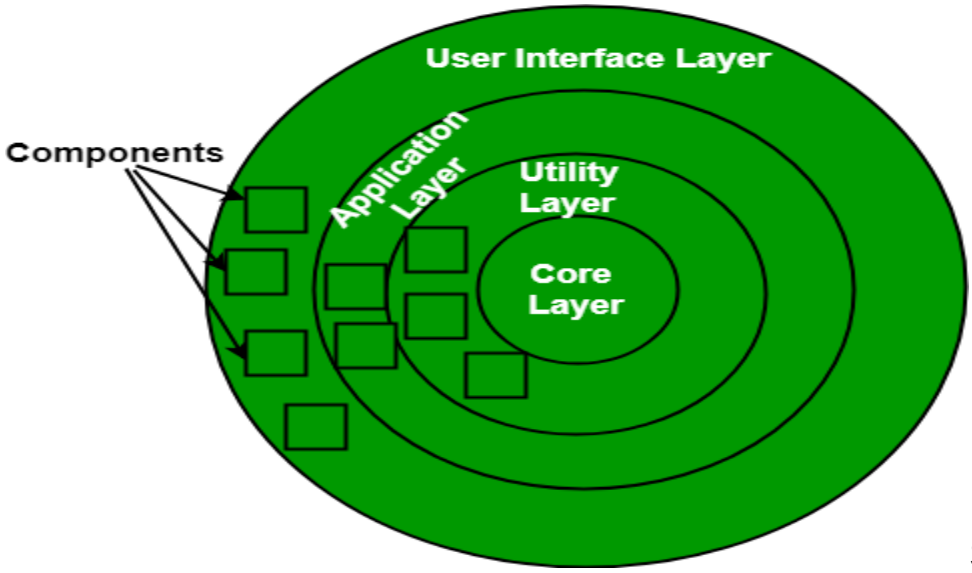
2. **Data flow architectures:**
   - This kind of architecture is used when input data to be transformed into output data through a series of computational manipulative components.

- 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 pipes.
- Pipes are used to transmit 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.



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

4. **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.
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 Engineering | User Interface Design

User interface is the front-end application view to which user interacts in order to use the software. The software becomes more popular if its user interface is:

- Attractive
- Simple to use
- Responsive in short time

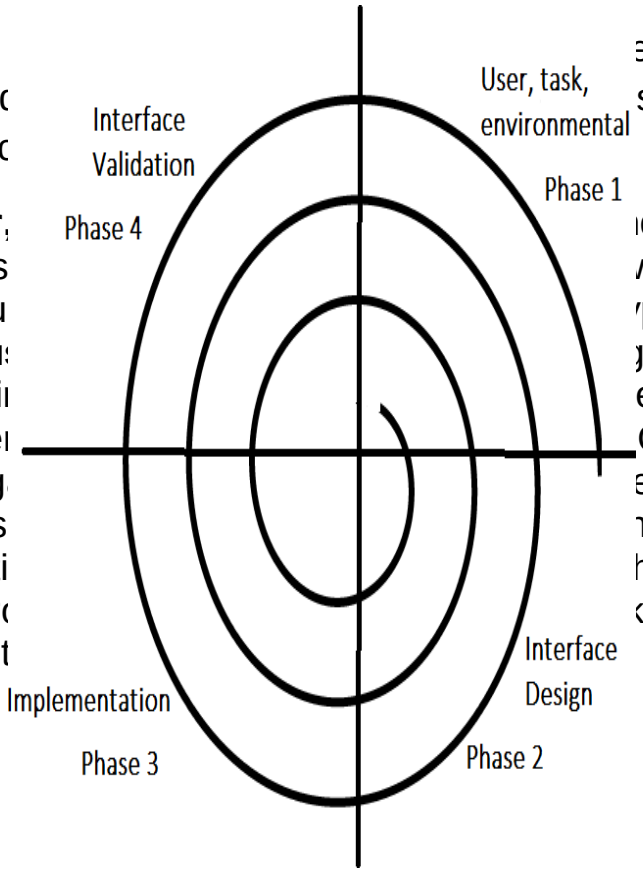
- Clear to understand
- Consistent on all interface

screens There are two types of User Interface:

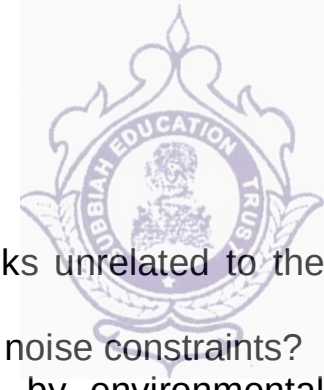
1. **Command Line Interface:** Command Line Interface provides a command prompt, where the user types the command and feeds to the system. The user needs to remember the syntax of the command and its use.
2. **Graphical User Interface:** Graphical User Interface provides the simple interactive interface to interact with the system. GUI can be a combination of both hardware and software. Using GUI, user interprets the software.

User Interface Design Process:

The analysis e is iterative and can be represented by a spiral moc ss of user interface consists of four framework ac



1. **User, d modeling:** Initially, the focus ill interact with the system, i.e. u pe of user, etc, based on the us gories. From each category requi e requirements developer under Once all the requirements are g ed. In the analysis part, the tasks he goals of the system are identi he analysis of the user envir environment. Among the quest asked are



- Where will the interface be located physically?
- Will the user be sitting, standing, or performing other tasks unrelated to the interface?
- Does the interface hardware accommodate space, light, or noise constraints?
- Are there special human factors considerations driven by environmental factors?

2. **Interface Design:** The goal of this phase is to define the set of interface objects and actions
   i.e. Control mechanisms that enable the user to perform desired tasks. Indicate how these control mechanisms affect the system. Specify the action sequence of tasks and subtasks, also called a user scenario. Indicate the state of the system when the user performs a particular task. Always follow the three golden rules stated by Theo Mandel. Design issues such as response time, command and action structure, error handling, and help facilities are considered as the design model is refined. This phase serves as the foundation for the implementation phase.
3. **Interface construction and implementation:** The implementation activity begins with the creation of prototype (model) that enables usage scenarios to be evaluated. As iterative design process continues a User Interface toolkit that allows the creation of windows, menus, device interaction, error messages, commands, and many other elements of an interactive environment can be used for completing the construction of an interface.
4. **Interface Validation:** This phase focuses on testing the interface. The interface should be in such a way that it should be able to perform tasks correctly and it should be able to
   handle a variety of tasks. It should achieve all the user's requirements. It should be easy to use and easy to learn. Users should accept the interface as a useful one in their work.

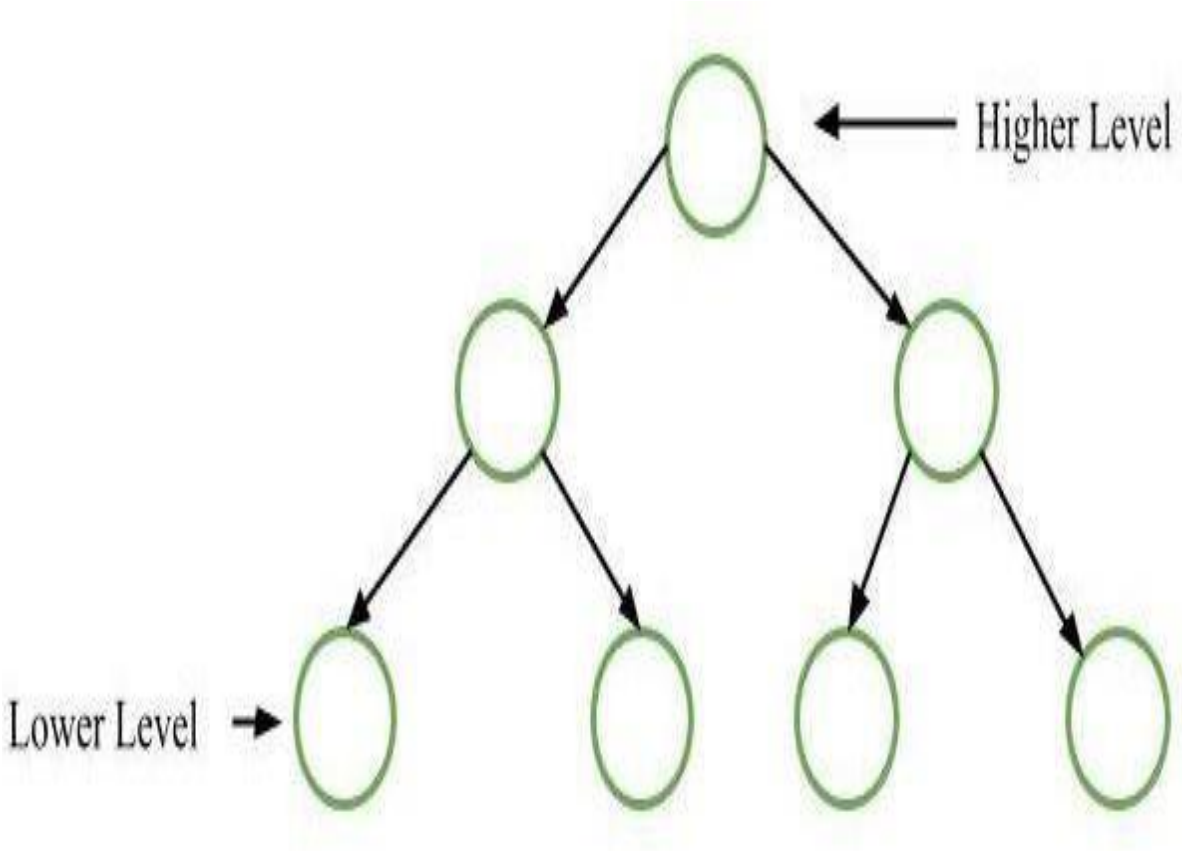
Software Engineering | Function Oriented Design

The design process for software systems often has two levels. At the first level the focus is on deciding which modules are needed for the system on the basis of SRS (Software Requirement Specification) and how the modules should be interconnected.

Function Oriented Design is an approach to software design where the design is decomposed into a set of interacting units where each unit has a clearly defined function.

Generic Procedure:

Start with a high level description of what the software / program does. Refine each part of the description one by one by specifying in greater details the functionality of each part. These points lead to Top-Down Structure.

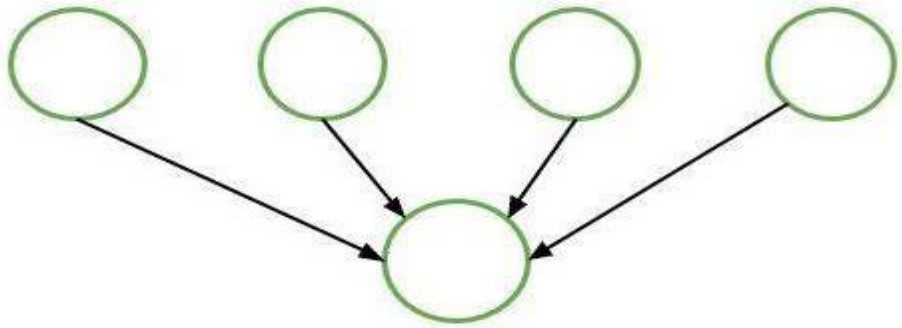


Problem in Top-Down design method:

Mostly each module is used by at most one other module and that module is called its Parent module.

Solution to the problem:

Designing of reusable module. It means modules use several modules to do their required functions.



Function Oriented Design Strategies:

Function Oriented Design Strategies are as follows:

1. **Data Flow Diagram (DFD):**
   A data flow diagram (DFD) maps out the flow of information for any process or system. It uses defined symbols like rectangles, circles and arrows, plus short text labels, to show data inputs, outputs, storage points and the routes between each destination.
2. **Data Dictionaries:**
   Data dictionaries are simply repositories to store information about all data items defined in
   DFDs. At the requirement stage, data dictionaries contains data items. Data dictionaries include Name of the item, Aliases (Other names for items), Description / purpose, Related data items, Range of values, Data structure definition / form.
3. **Structure Charts:**
   It is the hierarchical representation of system which partitions the system into black boxes (functionality is known to users but inner details are unknown). Components are read from top to bottom and left to right. When a module calls another, it views the called module as black box, passing required parameters and receiving results.

Pseudo Code:

Pseudo Code is system description in short English like phrases describing the function. It use keyword and indentation. Pseudo codes are used as replacement for flow charts. It decreases the amount of documentation required.

Consider a scenario of synchronous message passing. You have two components in your system that communicate with each other. Let's call the sender and receiver. The receiver asks for a service from the sender and the sender serves the request and waits for an acknowledgment from the receiver.

- There is another receiver that requests a service from the sender. The sender is blocked since it hasn't yet received any acknowledgment from the first receiver.
- The sender isn't able to serve the second receiver which can create problems. To solve this drawback, the Pub-Sub model was introduced.

What is Pub/Sub Architecture?

The Pub/Sub (Publisher/Subscriber) model is a messaging pattern used in software architecture to facilitate asynchronous communication between different components or systems. In this model, publishers produce messages that are then consumed by subscribers.

Key points of the Pub/Sub model include:

- **Publishers**: Entities that generate and send messages.
- **Subscribers**: Entities that receive and consume messages.
- **Topics**: Channels or categories to which messages are published.
- **Message Brokers**: Intermediaries that manage the routing of messages between publishers and subscribers.

Components of Pub/Sub Architecture?

In the Pub/Sub (Publish/Subscribe) model, there are several key components that work together to enable communication between publishers and subscribers. These components include:

1. Publisher

The Publisher is responsible for creating and sending messages to the Pub/Sub system. Publishers categorize messages into topics or channels based on their content. They do not need to know the identity of the subscribers.

2. Subscriber

The Subscriber is a recipient of messages in the Pub/Sub system. Subscribers express interest in receiving messages from specific topics.

They do not need to know the identity of the publishers. Subscribers receive messages from topics to which they are subscribed.

3. Topic

A Topic is a named channel or category to which messages are published. Publishers send messages to specific topics, and subscribers can subscribe to one or more topics to receive messages of interest. Topics help categorize messages and enable targeted message delivery to interested subscribers.

4. Message Broker

The Message Broker is an intermediary component that manages the routing of messages between publishers and subscribers. It receives messages from publishers and forwards them to subscribers based on their subscriptions. The Message Broker ensures that messages are delivered to the correct subscribers and can provide additional features such as message persistence, scalability, and reliability.

5. Message

A Message is the unit of data exchanged between publishers and subscribers in the Pub/Sub system. Messages can contain any type of data, such as text, JSON, or binary data. Publishers create messages and send them to the Pub/Sub system, and subscribers receive and process these messages.

6. Subscription

A Subscription represents a connection between a subscriber and a topic. Subscriptions define which messages a subscriber will receive based on the topics to which it is subscribed. Subscriptions can have different configurations, such as message delivery guarantees (e.g., at-most-once, at-least-once) and acknowledgment mechanisms.

What is Pipe and Filter Architecture?

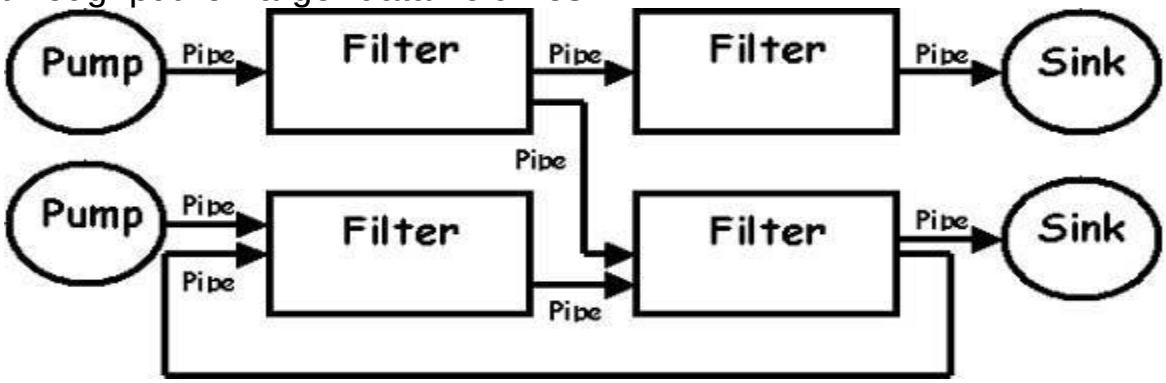
The Pipe and Filter architecture is a design pattern that divides a process into a series of distinct steps, called filters, linked by channels known as pipes. Each filter is dedicated to a particular processing function, whether it involves transforming, validating, or aggregating data. Data flows through these filters via pipes, which carry the output of one filter to the input of another.

- This architecture enhances modularity, as each filter operates independently and concentrates on a singular function. It also promotes reusability, allowing filters to be utilized across different systems or applications.
- Moreover, it is both flexible and scalable; filters can be added, removed, or rearranged with little effect on the overall system, and multiple instances of filters can function concurrently to manage larger data sets.

This organized structure makes the Pipe and Filter architecture a favored choice for tasks like data processing, compilers, and applications that require orderly and sequential data transformation.

Pipe and Filter Architecture

The architecture is highly modular, with each filter functioning independently, simplifying system comprehension, maintenance, and expansion. Filters can be added, removed, or rearranged without significant impact on the overall system, and parallel pipelines can boost throughput for larger data volumes.



- **Pumps:** These components initiate the process by acting as data sources, injecting data into the system and starting the flow through the pipeline.
- **Filters:** Each filter carries out a specific, standalone task, whether it be transforming, validating, or processing data before forwarding it. In the setup, there are two levels of filters; the first one processes data from the pump and hands it to the second filter, which continues the processing before passing it on.
- **Pipes:** Pipes serve as the channels for data movement between filters, linking each component in sequence and ensuring a smooth transfer of data. In diagrams, pipes are depicted as arrows connecting the components.

USER INTERFACE

The user interface is the **front-end application** view to which the **user interacts** to use the software. The software becomes more popular if its user interface is:

1. **Attractive**
2. **Simple to use**
3. **Responsive in a short time**
4. **Clear to understand**
5. **Consistent on all interface screens**

Types of User Interface

1. **Command Line Interface:** The Command Line Interface provides a command prompt, where the user types the command and feeds it to the system. The user needs to remember the syntax of the command and its use.
2. **Graphical User Interface:** Graphical User Interface provides a simple interactive interface to interact with the system. GUI can be a combination of both hardware and software. Using GUI, the user interprets the software.

User Interface Design Process

The **analysis** and **design** process of a user interface is iterative and can be represented by a **spiral model**. The analysis and design process of user interface consists of four framework activities.

1. User, Task, Environmental Analysis, and Modeling

Initially, the focus is based on the profile of users who will interact with the system, i.e., understanding, skill and knowledge, type of user, etc., based on the user's profile users are made into categories. From each category requirements are gathered. Based on the requirement's developer understand how to develop the interface. Once all the requirements are gathered a detailed analysis is conducted. In the analysis part, the tasks that the user performs to establish the goals of the system are identified, described and elaborated. The analysis of the user environment focuses on the physical work environment. Among the questions to be asked are:

1. Where will the interface be located physically?
2. Will the user be sitting, standing, or performing other tasks unrelated to the interface?
3. Does the interface hardware accommodate space, light, or noise constraints?
4. Are there special human factors considerations driven by environmental factors?2. Interface Design

The goal of this phase is to define the set of interface objects and actions i.e., control mechanisms that enable the user to perform desired tasks. Indicate how these control mechanisms affect the system. Specify the action sequence of tasks and subtasks, also called a user scenario. Indicate the state of the system when the user performs a particular task. Always follow the three golden rules stated by Theo Mandel. Design issues such as response time, command and action structure, error handling, and help facilities are considered as the design model is refined. This phase serves as the foundation for the implementation phase.


SUBBIAH EDUCATION TRUST

UNIT IV

SOFTWARE TESTING

Software testing is widely used technology because it is compulsory to test each and every software before deployment. Software testing such as Methods such as Black Box Testing, White Box Testing, Visual Box Testing and Gray Box Testing.

Software testing is a process of identifying the correctness of software by considering its all attributes (Reliability, Scalability, Portability, Re-usability, Usability) and evaluating the execution of software components to find the software bugs or errors or defects.

Types of Testing
Manual Testing Automation Testing
Types of Manual 3
White Box Testing Black Box Testing Grey Box Testing

1.White Box Techniques 5
Data Flow Testing
Control Flow Testing Branch Coverage Testing StatementCoverage Testing DecisionCoverage Testing

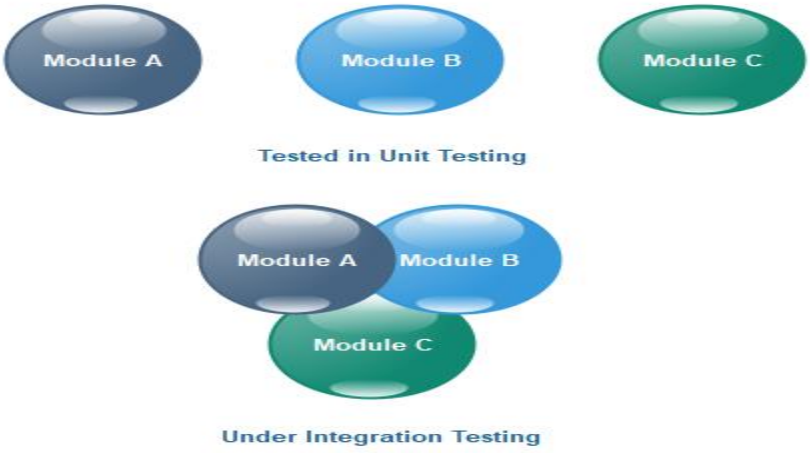
2.Black Box Techniques
Decision Table All-pair Testing Cause-Effect Testing State Transition Use Case

Types of Black Box 2
Functional Testing Non-Functional Testing
A)Types of Functional
Unit Testing Integration Testing System Testing
B) Types of Non-functional
Performance Testing Usability Testing Compatibility Testing

.

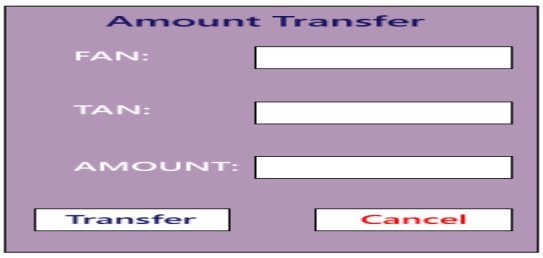
UNIT TESTING

Unit testing uses modules for testing purpose, and these modules are combined and tested in integration testing. The Software is developed with a number of software modules that are coded by different coders or programmers. The goal of integration testing is to check the correctness of communication among all the modules.

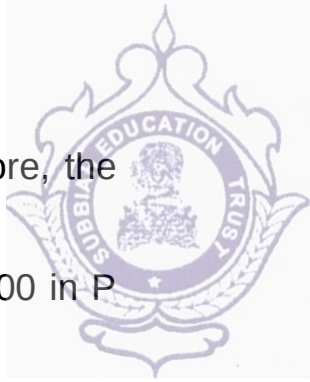


Once all the components or modules are working independently, then we need to check the data flow between the dependent modules is known as **integration testing**.

Let us see one sample example of a banking application, as we can see in the below image of amount transfer.



- First, we will login as a user **P** to amount transfer and send Rs200 amount, the confirmation message should be displayed on the screen as **amount transfer successfully**. Now logout as P and login as user **Q** and go to amount balance page and check for a balance in



that account = Present balance + Received Balance. Therefore, the integration test is successful.

- Also, we check if the amount of balance has reduced by Rs200 in P user account.
- Click on the transaction, in P and Q, the message should be displayed regarding the data and time of the amount transfer.

INTEGRATION TESTING

Integration testing is the process of testing the interface between two software units or modules. It focuses on determining the correctness of the interface. The purpose of integration testing is to expose faults in the interaction between integrated units. Once all the modules have been unit-tested, integration testing is performed.

Integration testing is a software testing technique that focuses on verifying the interactions and data exchange between different components or modules of a software application.

The goal of integration testing is to identify any problems or bugs that arise when different components are combined and interact with each other.

Integration testing is typically performed after unit testing and before system testing. It helps to identify and resolve integration issues early in the development cycle, reducing the risk of more severe and costly problems later on.

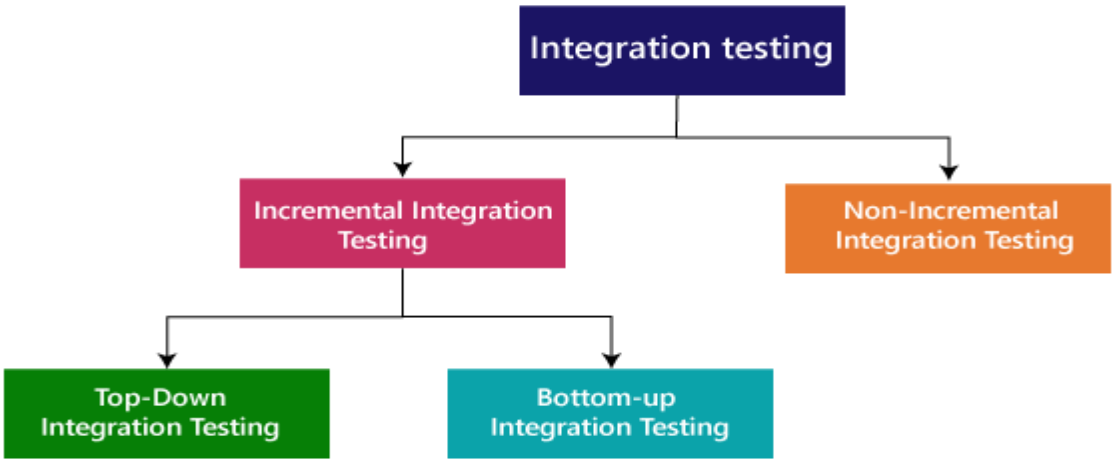
Integration testing can be done by picking module by module. This can be done so that there should be a proper sequence to be followed. And also if you don't want to miss out on any integration scenarios then you have to follow the proper sequence. Exposing the defects is the major focus of the integration testing and the time of interaction between the integrated units.

Integration testing is the second level of the software testing process comes after unit testing. In this testing, units or individual components of the software are tested in a group. The focus of the integration testing level is to expose defects at the time of interaction between integrated components or units.

Types of Integration Testing

Integration testing can be classified into two parts

- **Incremental integration testing**
- **Non-incremental integration testing**



Incremental Approach

In the Incremental Approach, modules are added in ascending order one by one or according to need. The selected modules must be logically related. Generally, two or more than two modules are added and tested to determine the correctness of functions. The process continues until the successful testing of all the modules.

For example: Suppose we have a Flipk art application, we will perform incremental integration testing, and the flow of the application would like this:

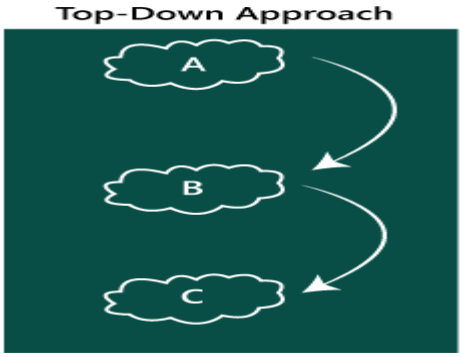
Flipkart→ Login→ Home → Search→ Add cart→Payment → Logout

Incremental integration testing is carried out by further methods:

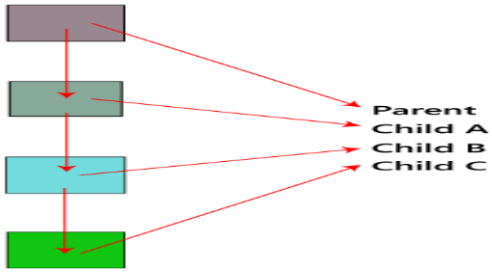
- Top-Down approach
- Bottom-Up approach

Top-Down Approach

The top-down testing strategy deals with the process in which higher level modules are tested with lower level modules until the successful completion of testing of all the modules. Major design flaws can be detected and fixed early because critical modules tested first. In this type of method, we will add the modules incrementally or one by one and check the data flow in the same order.



In the top-down approach, we will be ensuring that the module we are adding is the **child of the previous one like Child C is a child of Child B** and so on



Advantages:

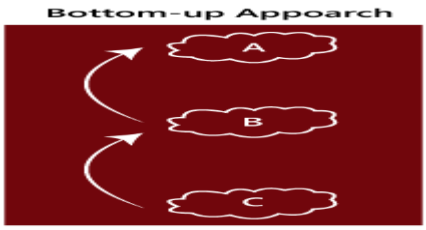
- Identification of defect is difficult.
- An early prototype is possible.

Disadvantages:

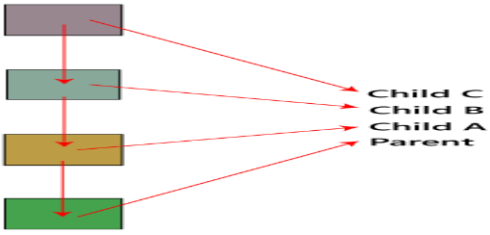
- Due to the high number of stubs, it gets quite complicated.
- Lower level modules are tested inadequately.
- Critical Modules are tested first so that fewer chances of defects.

Bottom-Up Method

The bottom to up testing strategy deals with the process in which lower level modules are tested with higher level modules until the successful completion of testing of all the modules. Top level critical modules are tested at last, so it may cause a defect. Or we can say that we will be adding the modules from **bottom to the top** and check the data flow in the same order.



In the bottom-up method, we will ensure that the modules we are adding **are the parent of the previous one** as we can see in the below image:



Advantages

- Identification of defect is easy.
- Do not need to wait for the development of all the modules as it saves time.

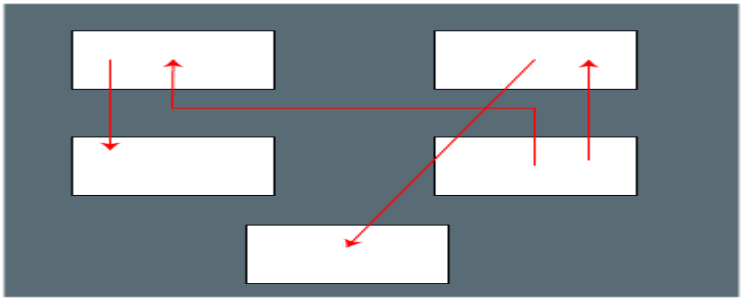
Disadvantages

- Critical modules are tested last due to which the defects can occur.
- There is no possibility of an early prototype.

NON- INCREMENTAL INTEGRATION TESTING

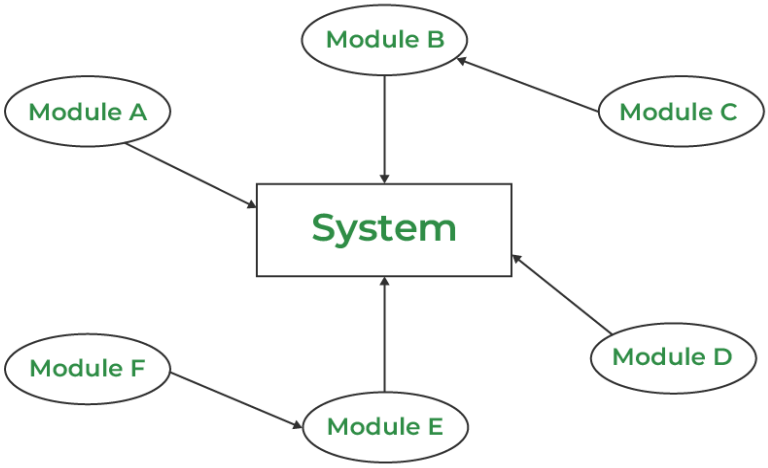
We will go for this method, when the data flow is very complex and when it is difficult to find who is a parent and who is a child. And in such case, we will create the data in any module bang on all other existing modules and

check if the data is present. Hence, it is also known as the **Big bang method**.



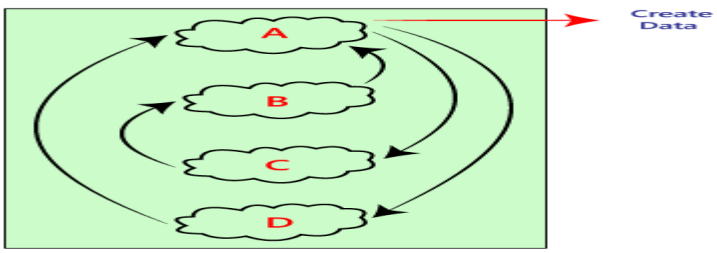
Big Bang Method

Big bang integration testing is a testing approach where all components or modules are integrated and tested as a single unit. This is done after all modules have been completed and before any system-level testing is performed



For example, consider a simple system with three modules A, B, and C. Module A has been tested and found to be working correctly. The same is true for modules B and C. To test the system as a whole, all three modules are integrated and tested together.

Since this testing can be done after completion of all modules due to that testing team has less time for execution of this process so that internally linked interfaces and high-risk critical modules can be missed easily.



Advantages:

- It is convenient for small size software systems.

Disadvantages:

- Identification of defects is difficult because finding the error where it came from is a problem, and we don't know the source of the bug.
- Small modules missed easily.
- Time provided for testing is very less.
- We may miss to test some of the interfaces.

REGRESSION TESTING

Regression testing is a black box testing techniques. It is used to authenticate a code change in the software does not impact the existing functionality of the product. Regression testing is making sure that the product works fine with new functionality, bug fixes, or any change in the existing feature.

Regression testing is a type of software testing. Test cases are re-executed to check the previous functionality of the application is working fine, and the new changes have not produced any bugs.

This method is used to test the product for modifications or any updates done. It ensures that any change in a product does not affect the existing module of the product. Verify that the bugs fixed and the newly added features not created any problem in the previous working version of the Software.

Regression tests are also known as the Verification Method. Test cases are often automated. Test cases are required to execute many times and running the same test case again and again manually, is time-consuming and tedious too.

How to perform Regression Testing?

The need for regression testing comes when software maintenance includes enhancements, error corrections, optimization, and deletion of existing features.

These modifications may affect system functionality. Regression Testing becomes necessary in this case.Regression testing can be performed using the following techniques:

1. Re-test All

Re-Test is one of the approaches to do regression testing. In this approach, all the test case suits should be re-executed. Here we can define re-test as when a test fails, and we determine the cause of the failure is a software fault. The fault is reported, we can expect a new version of the software in which defect fixed. In this case, we will need to execute the test again to confirm that the fault fixed. This is known as re-testing. Some will refer to this as confirmation testing.

The re-test is very expensive, as it requires enormous time and resources.

2. Regression test Selection

- In this technique, a selected test-case suit will execute rather than an entire test-case suit.
- The selected test case suits divided in two cases
    1. Reusable Test cases.
    2. Obsolete Test cases.
- Reusable test cases can use in succeeding regression cycle.
- Obsolete test cases can't use in succeeding regression cycle.

3. Prioritization of test cases

Prioritize the test case depending on business impact, critical and frequently functionality used. Selection of test cases will reduce the regression test suite.

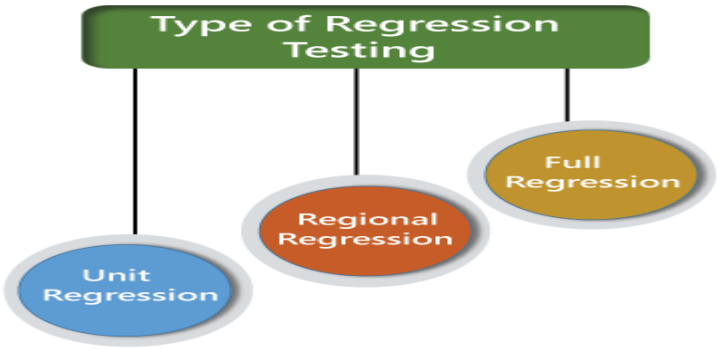
1. **When new functionality added to the application.**

2. **When there is a Change Requirement.**
3. **When the defect fixed**
4. **When there is a performance issue fix**
5. **When there is an environment change**

Types of Regression Testing

The different types of Regression Testing are as follows:

1. Unit Regression Testing [URT]
2. Regional Regression Testing[RRT]
3. Full or Complete Regression Testing [FRT]



1) Unit Regression Testing [URT]

In this, we are going to test only the changed unit, not the impact area, because it may affect the components of the same module.

2) Regional Regression testing [RRT]

In this, we are going to test the modification along with the impact area or regions, are called the **Regional Regression testing**. Here, we are testing the impact area because if there are dependable modules, it will affect the other modules also.

3) Full Regression testing [FRT]

During the second and the third release of the product, the client asks for adding 3-4 new features, and also some defects need to be fixed from the previous release. Then the testing team will do the Impact Analysis and identify that the above modification will lead us to test the entire product.

SYSTEM TESTING

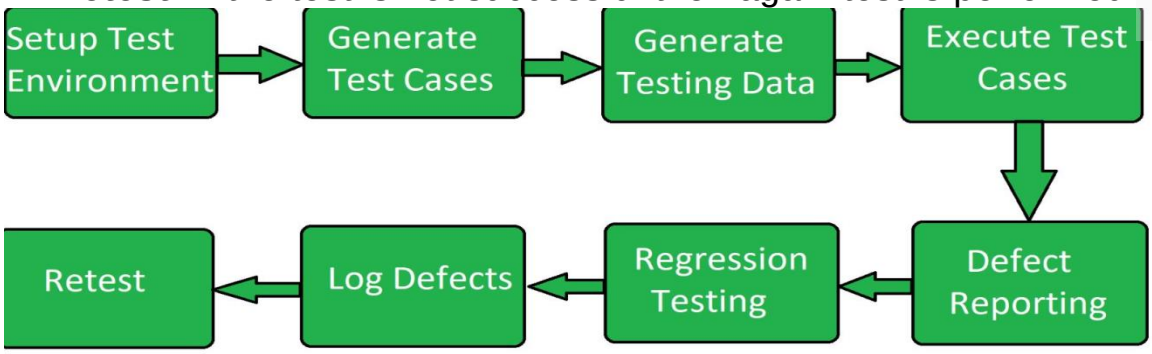
System testing is a type of software testing that evaluates the overall functionality and performance of a complete and fully integrated software solution. It tests if the system meets the specified requirements and if it is suitable for delivery to the end-users. This type of testing is performed after the integration testing and before the acceptance testing.

System Testing is a type of software testing that is performed on a complete integrated system to evaluate the compliance of the system with the corresponding requirements. In system testing, integration testing passed components are taken as input. The goal of integration testing is to detect any irregularity between the units that are integrated together. System testing detects defects within both the integrated units and the whole system. The result of system testing is the observed behavior of a component or a system when it is tested. **System Testing** is carried out on the whole system in the context of either system requirement specifications or functional requirement specifications or in the context of both. System testing tests the design and behavior of the system and also the expectations of the customer. It is performed to test the system beyond the bounds mentioned in the software requirements specification (SRS). System Testing is basically performed by a testing team that is independent of the development team that helps to test the quality of the system impartial. It has both functional and non-functional testing. **System Testing is a black-box testing**. System Testing is performed after the integration testing and before the acceptance testing.

System Testing Process: System Testing is performed in the following steps:

- **Test Environment Setup:** Create testing environment for the better quality testing.
- **Create Test Case:** Generate test case for the testing process.
- **Create Test Data:** Generate the data that is to be tested.
- **Execute Test Case:** After the generation of the test case and the test data, test cases are executed.

- **Defect Reporting:** Defects in the system are detected.
- **Regression Testing:** It is carried out to test the side effects of the testing process.
- **Log Defects:** Defects are fixed in this step.
- **Retest:** If the test is not successful then again test is performed.



Types of System Testing

- **Performance Testing:** Performance Testing is a type of software testing that is carried out to test the speed, scalability, stability and reliability of the software product or application.
- **Load Testing:** Load Testing is a type of software Testing which is carried out to determine the behavior of a system or software product under extreme load.
- **Stress Testing:** Stress Testing is a type of software testing performed to check the robustness of the system under the varying loads.
- **Scalability Testing:** Scalability Testing is a type of software testing which is carried out to check the performance of a software application or system in terms of its capability to scale up or scale down the number of user request load.

Tools used for System Testing

1. JMeter
2. Gallen Framework
3. Selenium
1. HP Quality Center/ALM
2. IBM Rational Quality Manager
3. Microsoft Test Manager
4. Selenium

Advantages of System Testing

- The testers do not require more knowledge of programming to carry out this testing.
- It will test the entire product or software so that we will easily detect the errors or defects which cannot be identified during the unit testing and integration testing.

- The testing environment is similar to that of the real time production or business environment.
- It checks the entire functionality of the system with different test scripts and also it covers the technical and business requirements of clients.
- After this testing, the product will almost cover all the possible bugs or errors and hence the development team will confidently go ahead with acceptance testing.

Here are some advantages of System Testing

- Verifies the overall functionality of the system.
- Detects and identifies system-level problems early in the development cycle.
- Helps to validate the requirements and ensure the system meets the user needs.
- Improves system reliability and quality.
- Facilitates collaboration and communication between development and testing teams.
- Enhances the overall performance of the system.
- Increases user confidence and reduces risks.
- Facilitates early detection and resolution of bugs and defects.
- Supports the identification of system-level dependencies and inter-module interactions.
- Improves the system’s maintainability and scalability.

SMOKE TESTING

Smoke Testing comes into the picture at the time of receiving build software from the development team. The purpose of smoke testing is to determine whether the build software is testable or not. It is done at the time of "building software." This process is also known as "Day 0".

It is a time-saving process. It reduces testing time because testing is done only when the key features of the application are not working or if the key bugs are not fixed. The focus of Smoke Testing is on the workflow of the core and primary functions of the application.

Testing the basic & critical feature of an application before doing one round of deep, rigorous testing (before checking all possible positive and negative values) is known as smoke testing.

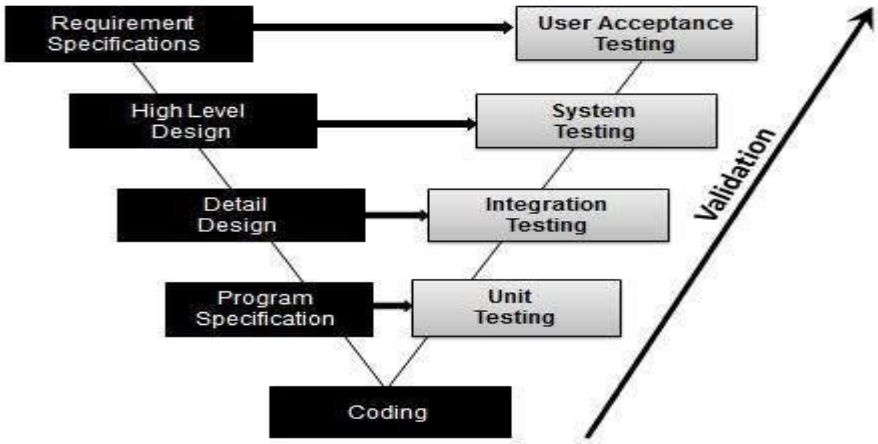
In the smoke testing, we only focus on the positive flow of the application and enter only valid data, not the invalid data. In smoke testing, we verify every build is testable or not; hence it is also known as **Build Verification Testing.**

Validation Testing ensures that the product actually meets the client's needs. It can also be defined as to demonstrate that the product fulfills its intended use when deployed on appropriate environment.

It answers to the question, Are we building the right product?

Validation Testing - Workflow:

Validation testing can be best demonstrated using V-Model. The Software/product under test is evaluated during this type of testing.



Activities:

- Unit Testing
- Integration Testing
- System Testing
- User Acceptance Testing

Standard Definition of Acceptance Testing

It is formal testing according to user needs, requirements, and business processes conducted to determine whether a system satisfies the acceptance criteria or not and to enable the users, customers, or other authorized entities to determine whether to accept the system or not.

Acceptance Testing is the last phase of software testing performed after System Testing and before making the system available for actual use.

Types of Acceptance Testing

1. User Acceptance Testing (UAT)

User acceptance testing is used to determine whether the product is working for the user correctly. Specific requirements which are quite often used by the customers are primarily picked for testing purposes. This is also termed as *End-User* Testing.

2. Business Acceptance Testing (BAT)

BAT is used to determine whether the product meets the business goals and purposes or not. BAT mainly focuses on business profits which are quite challenging due to the changing market conditions and new technologies, so the current implementation may have to being changed which results in extra budgets.

3. Contract Acceptance Testing (CAT)

CAT is a contract that specifies that once the product goes live, within a predetermined period, the acceptance test must be performed, and it should pass all the acceptance use cases..

4. Regulations Acceptance Testing (RAT)

RAT is used to determine whether the product violates the rules and regulations that are defined by the government of the country where it is being released. This may be unintentional but will impact negatively on the business. ctly responsible.

5. Operational Acceptance Testing (OAT)

OAT is used to determine the operational readiness of the product and is non-functional testing. It mainly includes testing of recovery, compatibility, maintainability, reliability, etc.

6. Alpha Testing

Alpha testing is used to determine the product in the development testing environment by a specialized testers team usually called alpha testers.

7. Beta Testing

Beta testing is used to assess the product by exposing it to the real end-users, typically called beta testers in their environment. Feedback is collected from the users and the defects are fixed. Also, this helps in enhancing the product to give a rich user experience.

WHITE BOX TESTING

The box testing approach of software testing consists of black box testing and white box testing. We are discussing here white box testing which also known as glass box is **testing, structural testing, clear box testing,**

open box testing and transparent box testing. It tests internal coding and infrastructure of a software focus on checking of predefined inputs against expected and desired outputs. It is based on inner workings of an application and revolves around internal structure testing. In this type of testing programming skills are required to design test cases. The primary goal of white box testing is to focus on the flow of inputs and outputs through the software and strengthening the security of the software.

The term 'white box' is used because of the internal perspective of the system. The clear box or white box or transparent box name denote the ability to see through the software's outer shell into its inner workings.

Developers do white box testing. In this, the developer will test every line of the code of the program. The developers perform the White-box testing and then send the application or the software to the testing team, where they will perform the black box testing and verify the application along with the requirements and identify the bugs and sends it to the developer.

The developer fixes the bugs and does one round of white box testing and sends it to the testing team. Here, fixing the bugs implies that the bug is deleted, and the particular feature is working fine on the application.

Here, the test engineers will not include in fixing the defects for the following reasons:

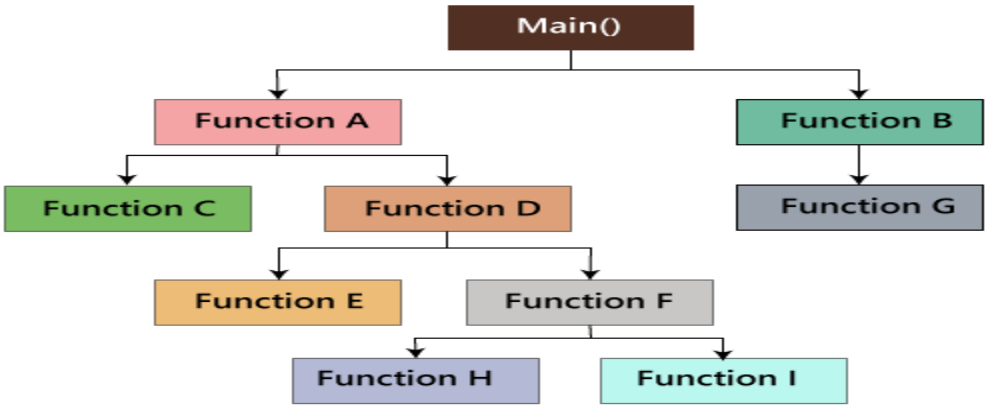
- Fixing the bug might interrupt the other features. Therefore, the test engineer should always find the bugs, and developers should still be doing the bug fixes.
- If the test engineers spend most of the time fixing the defects, then they may be unable to find the other bugs in the application.

The white box testing contains various tests, which are as follows:

- Path testing
- Loop testing
- Condition testing
- Testing based on the memory perspective
- Test performance of the program

Path testing

In the path testing, we will write the flow graphs and test all independent paths. Here writing the flow graph implies that flow graphs are representing the flow of the program and also show how every program is added with one another as we can see in the below image:



And test all the independent paths implies that suppose a path from main() to function G, first set the parameters and test if the program is correct in that particular path, and in the same way test all other paths and fix the bugs.

Loop testing

In the loop testing, we will test the loops such as while, for, and do-while, etc. and also check for ending condition if working correctly and if the size of the conditions is enough.

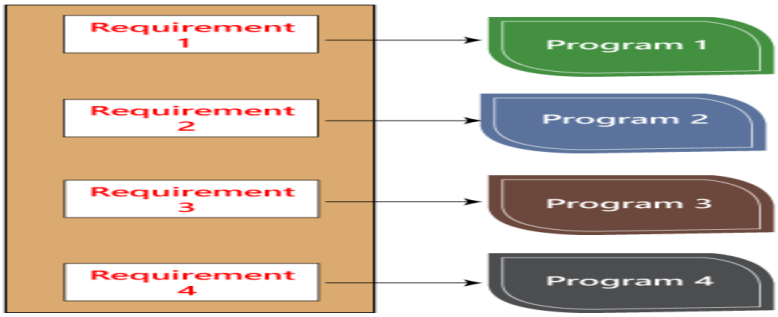
For example: we have one program where the developers have given about 50,000 loops.

1. {
2. while(50,000)
3. ……
4. ……
5. }

We cannot test this program manually for all the 50,000 loops cycle. So we write a small program that helps for all 50,000 cycles, as we can see in the below program, that test P is written in the similar language as the source code program, and this is known as a Unit test. And it is written by the developers only.

1. Test P
2. {
3. ……
4. …… }

As we can see in the below image that, we have various requirements such as 1, 2, 3, 4. And then, the developer writes the programs such as program 1,2,3,4 for the parallel conditions. Here the application contains the 100s line of codes.

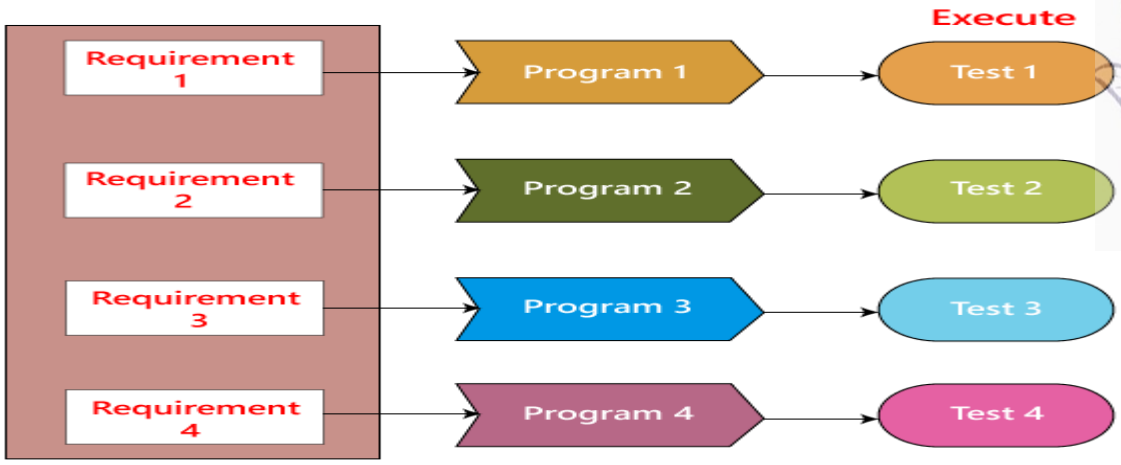


The developer will do the white box testing, and they will test all the five programs line by line of code to find the bug. If they found any bug in any of the programs, they will correct it. And they again have to test the system then this process contains lots of time and effort and slows down the product release time.

Now, suppose we have another case, where the clients want to modify the requirements, then the developer will do the required changes and test all four program again, which take lots of time and efforts.

These issues can be resolved in the following ways:

In this, we will write test for a similar program where the developer writes these test code in the related language as the source code. Then they execute these test code, which is also known as **unit test programs**. These test programs linked to the main program and implemented as programs.



Therefore, if there is any requirement of modification or bug in the code, then the developer makes the adjustment both in the main program and the test program and then executes the test program.

Condition testing

In this, we will test all logical conditions for both **true** and **false** values; that is, we will verify for both **if** and **else** condition.

For example:

1. if(condition) - true
2. {
3. …..
4. ……
5. ……
6. }
7. else - false
8. {
9. …..
10. ……
11. ……
12. }

The above program will work fine for both the conditions, which means that if the condition is accurate, and then else should be false and conversely.

Black-box testing is a type of software testing in which the tester is not concerned with the internal knowledge or implementation details of the software but rather focuses on validating the functionality based on the provided specifications or requirements.

Prerequisite - Software Testing | Basics

Black box testing can be done in the following ways:

1. Syntax-Driven Testing – This type of testing is applied to systems that can be syntactically represented by some language. For example, language can be represented by context-free grammar. In this, the test cases are generated so that each grammar rule is used at least once.

2. Equivalence partitioning – It is often seen that many types of inputs work similarly so instead of giving all of them separately we can group them and test only one input of each group. The idea is to partition the input domain of the system into several equivalence classes such that each member of the class works similarly, i.e., if a test case in one class results in some error, other members of the class would also result in the same error.

The technique involves two steps:

1. **Identification of equivalence class –** Partition any input domain into a minimum of two sets: **valid values** and **invalid values**. For example, if the valid range is 0 to 100 then select one valid input like 49 and one invalid like 104.
2. **Generating test cases –** (i) To each valid and invalid class of input assign a unique identification number. (ii) Write a test case covering all valid and invalid test cases considering that no two invalid inputs mask each other. To calculate the square root of a number, the equivalence classes will be **(a) Valid inputs:**
   - The whole number which is a perfect square-output will be an integer.
   - The entire number which is not a perfect square-output will be a decimal number.
   - Positive decimals
   - Negative numbers(integer or decimal).
   - Characters other than numbers like “a”,”!”,”;”, etc.

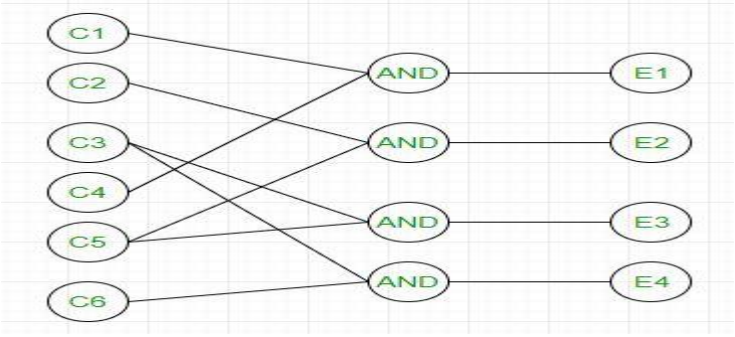
3. Boundary value analysis – Boundaries are very good places for errors to occur. Hence, if test cases are designed for boundary values of the input domain then the efficiency of testing improves and the probability of finding errors also increases. For example – If the valid range is 10 to 100 then test for 10,100 also apart from valid and invalid inputs.

4. Cause effect graphing – This technique establishes a relationship between logical input called causes with corresponding actions called the

effect. The causes and effects are represented using Boolean graphs.
The following steps are followed:

1. Identify inputs (causes) and outputs (effect).
2. Develop a cause-effect graph.
3. Transform the graph into a decision table.
4. Convert decision table rules to test cases.

For example, in the following cause-effect graph:



It can be converted into a decision table like:

		1	2	3	4
CAUSES	C1	1	0	0	0
	C2	0	1	0	0
	C3	0	0	1	1
	C4	1	0	0	0
	C5	0	1	1	0
	C6	0	0	0	1
EFFECTS	E1	x	-	-	-
	E2	-	x	-	-
	E3	-	-	x	-
	E4	-	-	-	x

Each column corresponds to a rule which will become a test case for testing. So there will be 4 test cases.

5. Requirement-based testing – It includes validating the requirements given in the SRS of a software system.

6. Compatibility testing – The test case results not only depends on the product but is also on the infrastructure for delivering functionality. When the infrastructure parameters are changed it is still expected to work properly. Some parameters that generally affect the compatibility of software are:

1. Processor (Pentium 3, Pentium 4) and several processors.
2. Architecture and characteristics of machine (32-bit or 64-bit).
3. Back-end components such as database servers.
4. Operating System (Windows, Linux, etc).

Black Box Testing Type

The following are the several categories of black box testing:

1. Functional Testing
2. Regression Testing
3. Nonfunctional Testing (NFT)

Functional Testing: It determines the system's software functional requirements.

Regression Testing: It ensures that the newly added code is compatible with the existing code. In other words, a new software update has no impact on the functionality of the software. This is carried out after a system maintenance operation and upgrades.
Nonfunctional Testing: Nonfunctional testing is also known as NFT. This testing is not functional testing of software. It focuses on the software's performance, usability, and scalability.

Tools Used for Black Box Testing:

1. Appium
2. Selenium
3. Microsoft Coded UI
4. Applitools
5. HP QTP.

What can be identified by Black Box Testing

1. Discovers missing functions, incorrect function & interface errors
2. Discover the errors faced in accessing the database
3. Discovers the errors that occur while initiating & terminating any functions.
4. Discovers the errors in performance or behaviour of software.

Features of black box testing:

1. **Independent testing:** Black box testing is performed by testers who are not involved in the development of the application, which helps to ensure that testing is unbiased and impartial.
2. **Testing from a user's perspective:** Black box testing is conducted from the perspective of an end user, which helps to ensure that the application meets user requirements and is easy to use.
3. **No knowledge of internal code:** Testers performing black box testing do not have access to the application's internal code, which allows them to focus on testing the application's external behaviour and functionality.
4. **Requirements-based testing:** Black box testing is typically based on the application's requirements, which helps to ensure that the application meets the required specifications.
5. **Different testing techniques:** Black box testing can be performed using various testing techniques, such as functional testing, usability testing, acceptance testing, and regression testing.
6. **Easy to automate:** Black box testing is easy to automate using various automation tools, which helps to reduce the overall testing time and effort.
7. **Scalability:** Black box testing can be scaled up or down depending on the size and complexity of the application being tested.
8. **Limited knowledge of application:** Testers performing black box testing have limited knowledge of the application being tested, which

helps to ensure that testing is more representative of how the end users will interact with the application.

Advantages of Black Box Testing:

- The tester does not need to have more functional knowledge or programming skills to implement the Black Box Testing.
- It is efficient for implementing the tests in the larger system.
- Tests are executed from the user's or client's point of view.
- Test cases are easily reproducible.
- It is used in finding the ambiguity and contradictions in the functional specifications.

Disadvantages of Black Box Testing:

- There is a possibility of repeating the same tests while implementing the testing process.
- Without clear functional specifications, test cases are difficult to implement.
- It is difficult to execute the test cases because of complex inputs at different stages of testing.
- Sometimes, the reason for the test failure cannot be detected.
- Some programs in the application are not tested.
- It does not reveal the errors in the control structure.
- Working with a large sample space of inputs can be exhaustive and consumes a lot of time.

UNIT V

SOFTWARE PROJECT MANAGEMENT

What is Project?

A project is a group of tasks that need to complete to reach a clear result. A project also defines as a set of inputs and outputs which are required to achieve a goal. Projects can vary from simple to difficult and can be operated by one person or a hundred.

SPM: Software Project Management (SPM) is a proper way of planning and leading software projects. It is a part of project management in which software projects are planned, implemented, monitored, and controlled.

there are three needs for software project management. These are:

1. Time
2. Cost
3. Quality

Need for Software Project Management

Software is a non-physical product. Software development is a new stream in business and there is very little experience in building software products. it is essential to manage software projects efficiently. It is necessary for an organization to deliver quality products, keep the cost within the client's budget constraint, and deliver the project as per schedule. Hence, in order, software project management is necessary to incorporate user requirements along with budget and time constraints.

Types of Management in SPM

1. Conflict Management

Conflict management is the process to restrict the negative features of conflict while increasing the positive features of conflict. The goal of conflict management is to improve learning and group results including efficacy or performance in an organizational setting. Properly managed conflict can enhance group results.

2. Risk Management

Risk management is the analysis and identification of risks that is followed by synchronized and economical implementation of resources to minimize, operate and control the possibility or effect of unfortunate events or to maximize the realization of opportunities.

3. Requirement Management

It is the process of analyzing, prioritizing, tracking, and documenting requirements and then supervising change and communicating to pertinent stakeholders. It is a continuous process during a project.

4. Change Management

Change management is a systematic approach to dealing with the transition or transformation of an organization's goals, processes, or technologies. The purpose of change management is to execute strategies for effecting change, controlling change, and helping people to adapt to change.

5. Software Configuration Management

Software configuration management is the process of controlling and tracking changes in the software, part of the larger cross-disciplinary field of configuration management. Software configuration management includes revision control and the inauguration of baselines.

6. Release Management

Release Management is the task of planning, controlling, and scheduling the built-in deploying releases. Release management ensures that the organization delivers new and enhanced services required by the customer while protecting the integrity of existing services.

Aspects of Software Project Management

The list of focus areas it can tackle and the broad upsides of Software Project Management is:

1. Planning

The software project manager lays out the complete project's blueprint. The project plan will outline the scope, resources, timelines, techniques, strategy, communication, testing, and maintenance steps. SPM can aid greatly here.

2. Leading

A software project manager brings together and leads a team of engineers, strategists, programmers, designers, and data scientists. Leading a team necessitates exceptional communication, interpersonal, and leadership abilities. One can only hope to do this effectively if one sticks with the core SPM principles.

3. Execution

SPM comes to the rescue here also as the person in charge of software projects (if well versed with SPM/Agile methodologies) will ensure that each stage of the project is completed successfully. measuring progress, monitoring to check how teams function, and generating status reports are all part of this process.

4. Time Management

Abiding by a timeline is crucial to completing deliverables successfully. This is especially difficult when managing software projects because changes to the original project charter are unavoidable over time. To assure progress in the face of blockages or changes, software project managers ought to be specialists in managing risk and emergency preparedness. This Risk Mitigation and management is one of the core tenets of the philosophy of SPM.

5. Budget

Software project managers, like conventional project managers, are responsible for generating a project budget and adhering to it as closely as feasible, regulating spending, and reassigning funds as needed. SPM teaches us how to effectively manage the monetary aspect of projects to avoid running into a financial crunch later on in the project.

6. Maintenance

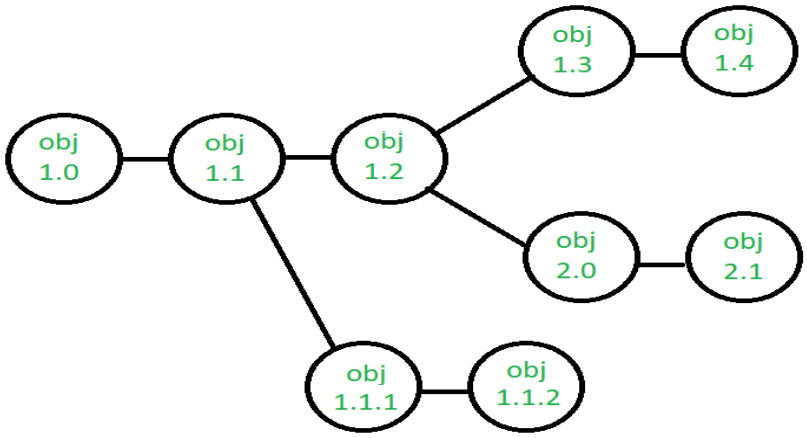
Software project management emphasizes continuous product testing to find and repair defects early, tailor the end product to the needs of the client, and keep the project on track. The software project manager makes ensuring that the product is thoroughly tested, analyzed, and adjusted as needed. Another point in favor of SPM.

SOFTWARE CONFIGURATION MANAGEMENT

System Configuration Management (SCM) is an arrangement of exercises that controls change by recognizing the items for change, setting up connections between those things, making/characterizing instruments for overseeing diverse variants, controlling the changes being executed in the current framework, inspecting and revealing/reporting on the changes made. It is essential to control the changes because if the changes are not checked legitimately then they may wind up undermining a well-run programming. In this way, SCM is a fundamental piece of all project management activities.

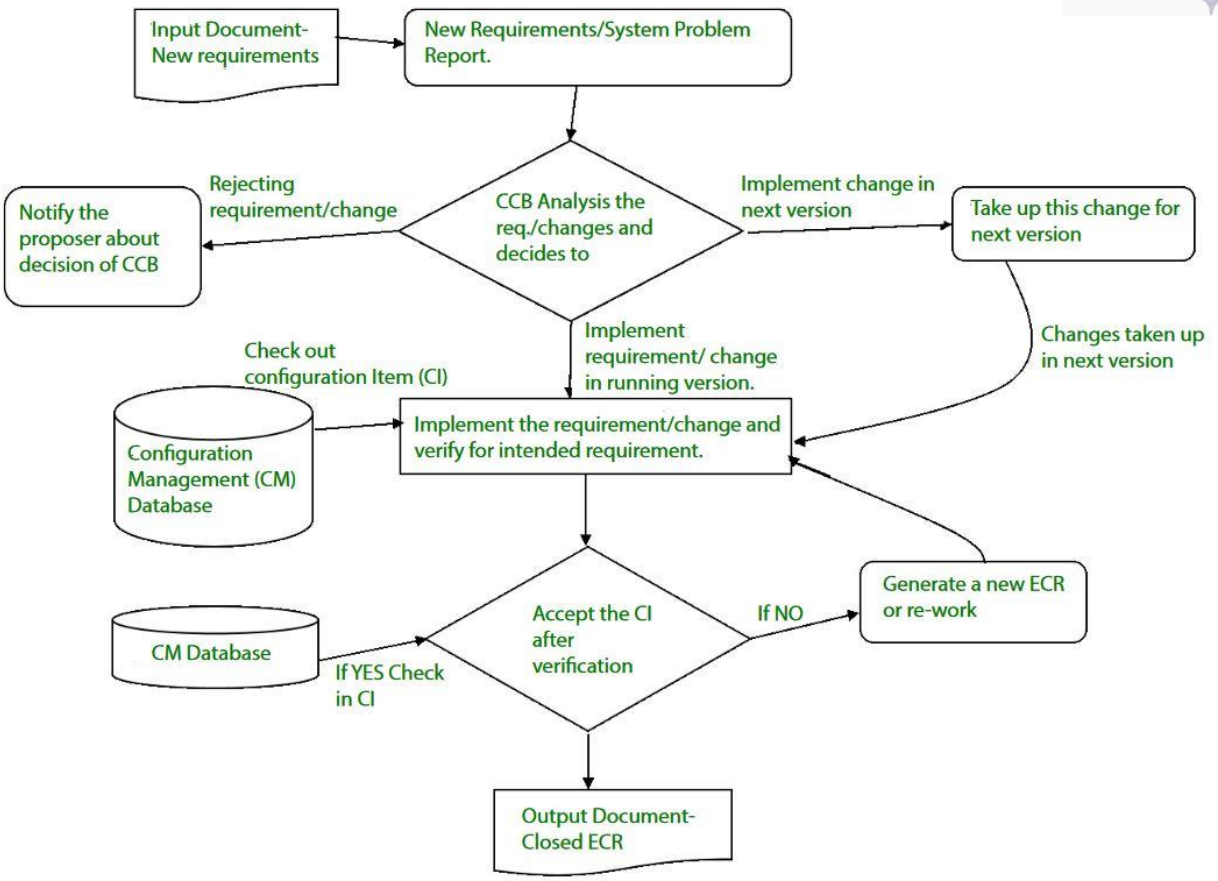
Processes involved in SCM – Configuration management provides a disciplined environment for smooth control of work products. It involves the following activities:

1. Identification and Establishment – Identifying the configuration items from products that compose baselines at given points in time (a baseline is a set of mutually consistent Configuration Items, which has been formally reviewed and agreed upon, and serves as the basis of further development). Establishing relationships among items, creating a mechanism to manage multiple levels of control and procedure for the change management system.
2. Version control – Creating versions/specifications of the existing product to build new products with the help of the SCM system. A description of the version is given below:



3. Suppose after some changes, the version of the configuration object changes from 1.0 to 1.1. Minor corrections and changes result in versions 1.1.1 and 1.1.2, which is followed by a major update that is object 1.2. The development of object 1.0 continues through 1.3 and 1.4, but finally, a noteworthy change to the object results in a new evolutionary path, version 2.0. Both versions are currently supported.
3. Change control – Controlling changes to Configuration items (CI). The change control process is explained in Figure below:
4. A change request (CR) is submitted and evaluated to assess technical merit, potential side effects, the overall impact on other configuration objects and system functions, and the projected cost of the change. The results of the evaluation are presented as a change report, which is used by a change control board (CCB) —a person or group who makes a final decision on the status and priority of the change. An engineering change Request (ECR) is generated for each approved change. Also, CCB notifies the developer in case the change is rejected with proper reason. The ECR describes the change to be made, the constraints that must be respected, and the criteria for review and audit. The object to be changed is “checked out” of the project database, the change is made, and then the object is tested

again. The object is then “checked in” to the database and appropriate version control mechanisms are used to create the next version of the software.



4. Configuration auditing – A software configuration audit complements the formal technical review of the process and product. It focuses on the technical correctness of the configuration object that has been modified. The audit confirms the completeness, correctness, and consistency of items in the SCM system and tracks action items from the audit to closure.
5. Reporting – Providing accurate status and current configuration data to developers, testers, end users, customers, and stakeholders through admin guides, user guides, FAQs, Release notes, Memos, Installation Guide, Configuration guides, etc.

System Configuration Management (SCM) is a software engineering practice that focuses on managing the configuration of software systems and ensuring that software components are properly controlled, tracked, and stored. It is a critical aspect of software development, as it helps to ensure that changes made to a software system are properly coordinated and that the system is always in a known and stable state.

SCM involves a set of processes and tools that help to manage the different components of a software system, including source code, documentation, and other assets. It enables teams to track changes made to the software system, identify when and why changes were made, and manage the integration of these changes into the final product.

Importance of Software Configuration Management

1. Effective Bug Tracking: Linking code modifications to issues that have been reported, makes bug tracking more effective.
2. Continuous Deployment and Integration: SCM combines with continuous processes to automate deployment and testing, resulting in more dependable and timely software delivery.
3. Risk management: SCM lowers the chance of introducing critical flaws by assisting in the early detection and correction of problems.
4. Support for Big Projects: Source Code Control (SCM) offers an orderly method to handle code modifications for big projects, fostering a well-organized development process.
5. Reproducibility: By recording precise versions of code, libraries, and dependencies, source code versioning (SCM) makes builds repeatable.
6. Parallel Development: SCM facilitates parallel development by enabling several developers to collaborate on various branches at once.

The main advantages of SCM

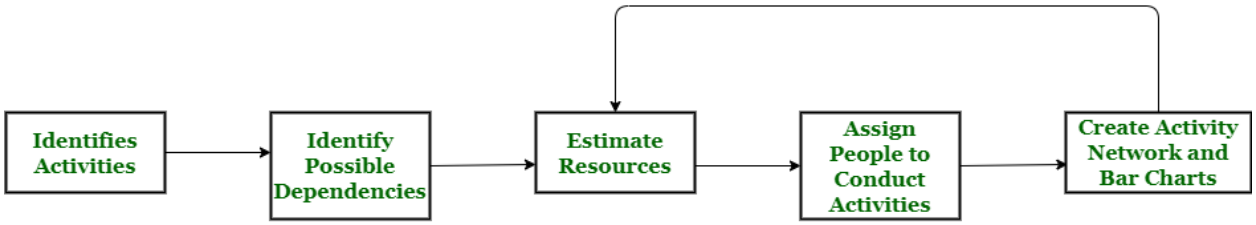
1. Improved productivity and efficiency by reducing the time and effort required to manage software changes.
2. Reduced risk of errors and defects by ensuring that all changes were properly tested and validated.
3. Increased collaboration and communication among team members by providing a central repository for software artifacts.
4. Improved quality and stability of software systems by ensuring that all changes are properly controlled and managed.

The main disadvantages of SCM

1. Increased complexity and overhead, particularly in large software systems.
2. Difficulty in managing dependencies and ensuring that all changes are properly integrated.
3. Potential for conflicts and delays, particularly in large development teams with multiple contributors.

PROJECT SCHEDULEING

Project schedule simply means a mechanism that is used to communicate and know about that tasks are needed and has to be done or performed and which organizational resources will be given or allocated to these tasks and in what time duration or time frame work is needed to be performed. Effective project scheduling leads to success of project, reduced cost, and increased customer satisfaction. Scheduling in project management means to list out activities, deliverables, and milestones within a project that are delivered. It contains more notes than your average weekly planner notes. The most common and important form of project schedule is Gantt chart.



Project Scheduling Process

PROJECT SIZE ESTIMATION TECHNIQUES – SOFTWARE ENGINEERING

Project size estimation is a crucial aspect of software engineering, as it helps in planning and allocating resources for the project. Here are some of the popular project size estimation techniques used in software engineering:

- Expert Judgment: In this technique, a group of experts in the relevant field estimates the project size based on their experience and expertise. This technique is often used when there is limited information available about the project.
- Analogous Estimation: This technique involves estimating the project size based on the similarities between the current project and previously completed projects. This technique is useful when historical data is available for similar projects.

- Bottom-up Estimation: In this technique, the project is divided into smaller modules or tasks, and each task is estimated separately. The estimates are then aggregated to arrive at the overall project estimate.
- Three-point Estimation: This technique involves estimating the project size using three values: optimistic, pessimistic, and most likely. These values are then used to calculate the expected project size using a formula such as the PERT formula.
- Function Points: This technique involves estimating the project size based on the functionality provided by the software. Function points consider factors such as inputs, outputs, inquiries, and files to arrive at the project size estimate.
- Use Case Points: This technique involves estimating the project size based on the number of use cases that the software must support. Use case points consider factors such as the complexity of each use case, the number of actors involved, and the number of use cases.
- Parametric Estimation: For precise size estimation, mathematical models founded on project parameters and historical data are used.
- COCOMO (Constructive Cost Model): It is an algorithmic model that estimates effort, time, and cost in software development projects by taking into account a number of different elements.
- Wideband Delphi: Consensus-based estimating method for balanced size estimations that combines expert estimates from anonymous experts with cooperative conversations.
- Monte Carlo simulation: This technique, which works especially well for complicated and unpredictable projects, estimates project size and analyses hazards using statistical methods and random sampling.

Each of these techniques has its strengths and weaknesses, and the choice of technique depends on various factors such as the project's complexity, available data, and the expertise of the team.

Importance of Project Size Estimation Techniques

- Resource Allocation: Appropriate distribution of financial and human resources is ensured by accurate estimation.
- Risk management: Early risk assessment helps with mitigation techniques by taking into account the complexity of the project.
- Time management: Facilitates the creation of realistic schedules and milestones for efficient time management.
- Cost control and budgeting: Both the terms are closely related, which lowers the possibility of cost overruns.
- Resource Allocation: Enables efficient task delegation and work allocation optimization.

- Scope Definition: Defines the scope of a project, keeps project boundaries intact and guards against scope creep.

Estimating the size of the Software

Estimation of the size of the software is an essential part of Software Project Management. It helps the project manager to further predict the effort and time that will be needed to build the project. Various measures are used in project size estimation. Some of these are:

- Lines of Code
- Number of entities in the ER diagram
- Total number of processes in detailed data flow diagram
- Function points
- KLOC- Thousand lines of code
- NLOC- Non-comment lines of code
- KDSI- Thousands of delivered source instruction

The size is estimated by comparing it with the existing systems of the same kind. The experts use it to predict the required size of various components of software and then add them to get the total size.

It's tough to estimate LOC by analyzing the problem definition. Only after the whole code has been developed can accurate LOC be estimated. This statistic is of little utility to project managers because project planning must be completed before development activity can begin.

Two separate source files having a similar number of lines may not require the same effort. A file with complicated logic would take longer to create than one with simple logic. Proper estimation may not be attainable based on LOC.

The length of time it takes to solve an issue is measured in LOC. This statistic will differ greatly from one programmer to the next. A seasoned programmer can write the same logic in fewer lines than a newbie coder.

Advantages

- Universally accepted and is used in many models like COCOMO.
- Estimation is closer to the developer's perspective.
- Both people throughout the world utilize and accept it.
- At project completion, LOC is easily quantified.
- It has a specific connection to the result.
- Simple to use.

Disadvantages:

- Different programming languages contain a different number of lines.
- No proper industry standard exists for this technique.
- It is difficult to estimate the size using this technique in the early stages of the project.

- When platforms and languages are different, LOC cannot be used to normalize.

2. Number of entities in ER diagram: ER model provides a static view of the project. It describes the entities and their relationships. The number of entities in ER model can be used to measure the estimation of the size of the project. The number of entities depends on the size of the project. This is because more entities needed more classes/structures thus leading to more coding.

Advantages:

- Size estimation can be done during the initial stages of planning.
- The number of entities is independent of the programming technologies used.

Disadvantages:

- No fixed standards exist. Some entities contribute more to project size than others.
- Just like FPA, it is less used in the cost estimation model. Hence, it must be converted to LOC.

3. Total number of processes in detailed data flow diagram: Data Flow Diagram(DFD) represents the functional view of software. The model depicts the main processes/functions involved in software and the flow of data between them. Utilization of the number of functions in DFD to predict software size. Already existing processes of similar type are studied and used to estimate the size of the process. Sum of the estimated size of each process gives the final estimated size.

Advantages:

- It is independent of the programming language.
- Each major process can be decomposed into smaller processes. This will increase the accuracy of the estimation.

Disadvantages:

- Studying similar kinds of processes to estimate size takes additional time and effort.
- All software projects are not required for the construction of DFD.

4. Function Point Analysis: In this method, the number and type of functions supported by the software are utilized to find FPC(function point count). The steps in function point analysis are:

- Count the number of functions of each proposed type.
- Compute the Unadjusted Function Points(UFP).
- Find the Total Degree of Influence(TDI).
- Compute Value Adjustment Factor(VAF).
- Find the Function Point Count(FPC).

The explanation of the above points is given below:

- Count the number of functions of each proposed type: Find the number of functions belonging to the following types:

External Inputs: Functions related to data entering the system.
External outputs: Functions related to data exiting the system.
External Inquiries: They lead to data retrieval from the system but don't change the system.
Internal Files: Logical files maintained within the system. Log files are not included here.
External interface Files: These are logical files for other applications which are used by our system.

- Compute the Unadjusted Function Points(UFP): Categorise each of the five function types like simple, average, or complex based on their complexity. Multiply the count of each function type with its weighting factor and find the weighted sum. The weighting factors for each type based on their complexity are as follows:

Function type	Simple	Average	Complex
External Inputs	3	4	6
External Output	4	5	7
External Inquiries	3	4	6
Internal Logical Files	7	10	15
External Interface Files	5	7	10

- Find Total Degree of Influence: Use the '14 general characteristics' of a system to find the degree of influence of each of them. The sum of all 14 degrees of influence will give the TDI. The range of TDI is 0 to 70. The 14 general characteristics are: Data Communications, Distributed Data Processing, Performance, Heavily Used

Configuration, Transaction Rate, On-Line Data Entry, End-user Efficiency, Online Update, Complex Processing Reusability, Installation Ease, Operational Ease, Multiple Sites and Facilitate Change.

Each of the above characteristics is evaluated on a scale of 0-5.

- Compute Value Adjustment Factor(VAF): Use the following formula to calculate VAF

VAF = (TDI * 0.01) + 0.65

- Find the Function Point Count: Use the following formula to calculate FPC

FPC = UFP * VAF

Advantages:

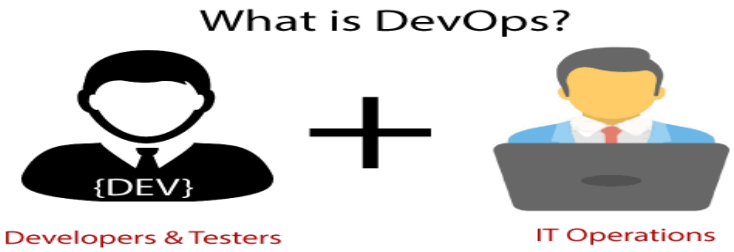
- It can be easily used in the early stages of project planning.
- It is independent of the programming language.
- It can be used to compare different projects even if they use different technologies(database, language, etc).
- Disadvantages:
- It is not good for real-time systems and embedded systems.
- Many cost estimation models like COCOMO use LOC and hence FPC must be converted to LOC.

DEVOPS TUTORIAL

The DevOps is the combination of two words, one is Development and other is Operations. It is a culture to promote the development and operation process collectively.The DevOps tutorial will help you to learn DevOps basics and provide depth knowledge of various DevOps tools such as Git, Ansible, Docker, Puppet, Jenkins, Chef, Nagios, and Kubernetes.

What is DevOps?

The DevOps is a combination of two words, one is software Development, and second is Operations. This allows a single team to handle the entire application lifecycle, from development to testing, deployment, and operations. DevOps helps you to reduce the disconnection between software developers, quality assurance (QA) engineers, and system administrators.



DevOps promotes collaboration between Development and Operations team to deploy code to production faster in an automated & repeatable way.DevOps helps to increase organization speed to deliver applications and services.

It also allows organizations to serve their customers better and compete more strongly in the market. With the help of DevOps, quality, and speed of the application delivery has improved to a great extent.

DevOps is all about the integration of the operations and development process. Organizations that have adopted DevOps noticed a 22% improvement in software quality and a 17% improvement in application deployment frequency and achieve a 22% hike in customer satisfaction. 19% of revenue hikes as a result of the successful DevOps implementation.

Why DevOps?

Before going further, we need to understand why we need the DevOps over the other methods.

- The operation and development team worked in complete isolation.
- After the design-build, the testing and deployment are performed respectively. That's why they consumed more time than actual build cycles.

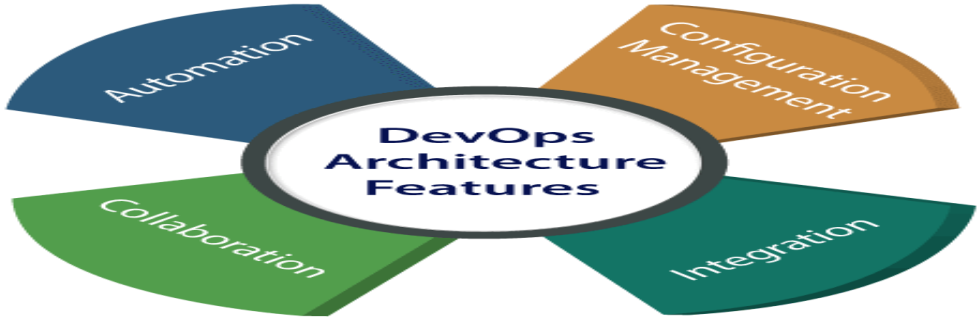
- Without the use of DevOps, the team members are spending a large amount of time on designing, testing, and deploying instead of building the project.
- Manual code deployment leads to human errors in production.
- Coding and operation teams have their separate timelines and are not in synch, causing further delays.

DevOps History

- In 2009, the first conference named DevOpsdays was held in Ghent Belgium. Belgian consultant and Patrick Debois founded the conference.
- In 2012, the state of DevOps report was launched and conceived by Alanna Brown at Puppet.
- In 2014, the annual State of DevOps report was published by Nicole Forsgren, Jez Humble, Gene Kim, and others. They found DevOps adoption was accelerating in 2014 also.
- In 2015, Nicole Forsgren, Gene Kim, and Jez Humble founded DORA (DevOps Research and Assignment).
- In 2017, Nicole Forsgren, Gene Kim, and Jez Humble published "Accelerate: Building and Scaling High Performing Technology Organizations".

DevOps Architecture Features

Here are some key features of DevOps architecture, such as:



1) Automation

Automation can reduce time consumption, especially during the testing and deployment phase. The productivity increases, and releases are made quicker by automation. This will lead in catching bugs quickly so that it can be fixed easily. For contiguous delivery, each code is defined through automated tests, cloud-based services, and builds. This promotes production using automated deploys.

2) Collaboration

The Development and Operations team collaborates as a DevOps team, which improves the cultural model as the teams become more productive with their productivity, which strengthens accountability and ownership. The teams share their responsibilities and work closely in sync, which in turn makes the deployment to production faster.

3) Integration

Applications need to be integrated with other components in the environment. The integration phase is where the existing code is combined with new functionality and then tested. Continuous integration and testing enable continuous development. The frequency in the releases and micro-services leads to significant operational challenges. To overcome such problems, continuous integration and delivery are implemented to deliver in a quicker, safer, and reliable manner.

4) Configuration management

It ensures the application to interact with only those resources that are concerned with the environment in which it runs. The configuration files are not created where the external configuration to the application is separated from the source code. The configuration file can be written during deployment, or they can be loaded at the run time, depending on the environment in which it is running.

Advantages

- DevOps is an excellent approach for quick development and deployment of applications.
- It responds faster to the market changes to improve business growth.

- DevOps escalate business profit by decreasing software delivery time and transportation costs.
- DevOps clears the descriptive process, which gives clarity on product development and delivery.
- It improves customer experience and satisfaction.
- DevOps simplifies collaboration and places all tools in the cloud for customers to access.
- DevOps means collective responsibility, which leads to better team engagement and productivity.

Disadvantages

- DevOps professional or expert's developers are less available.
- Developing with DevOps is so expensive.
- Adopting new DevOps technology into the industries is hard to manage in short time.

DevOps Tools

Here are some most popular DevOps tools with brief explanation shown in the below image, such as:

1) Puppet

Puppet is the most widely used DevOps tool. It allows the delivery and release of the technology changes quickly and frequently. It has features of versioning, automated testing, and continuous delivery. It enables to

manage entire infrastructure as code without expanding the size of the team.

Features

- Real-time context-aware reporting.
- Model and manage the entire environment.
- Defined and continually enforce infrastructure.
- Desired state conflict detection and remediation.
- It inspects and reports on packages running across the infrastructure.
- It eliminates manual work for the software delivery process.
- It helps the developer to deliver great software quickly.

2) Ansible

Ansible is a leading DevOps tool. Ansible is an open-source IT engine that automates application deployment, cloud provisioning, intra service orchestration, and other IT tools. It makes it easier for DevOps teams to scale automation and speed up productivity.

Ansible is easy to deploy because it does not use any agents or custom security infrastructure on the client-side, and by pushing modules to the clients. These modules are executed locally on the client-side, and the output is pushed back to the Ansible server.

Features

- It is easy to use to open source deploy applications.
- It helps in avoiding complexity in the software development process.
- It eliminates repetitive tasks.
- It manages complex deployments and speeds up the development process.

3) Docker

Docker is a high-end DevOps tool that allows building, ship, and run distributed applications on multiple systems. It also helps to assemble the

apps quickly from the components, and it is typically suitable for container management.

Features

- It configures the system more comfortable and faster.
- It increases productivity.
- It provides containers that are used to run the application in an isolated environment.
- It routes the incoming request for published ports on available nodes to an active container. It allows saving secrets into the swarm itself.

4) Nagios

Nagios is one of the more useful tools for DevOps. It can determine the errors and rectify them with the help of network, infrastructure, server, and log monitoring systems.

Features

- It provides complete monitoring of desktop and server operating systems.
- The network analyzer helps to identify bottlenecks and optimize bandwidth utilization.
- It helps to monitor components such as services, application, OS, and network protocol.
- It also provides to complete monitoring of Java Management Extensions.

5) CHEF

A chef is a useful tool for achieving scale, speed, and consistency. The chef is a cloud-based system and open source technology. This technology uses Ruby encoding to develop essential building blocks such as recipes and cookbooks. The chef is used in infrastructure automation and helps in reducing manual and repetitive tasks for infrastructure management.

Chef has got its convention for different building blocks, which are required to manage and automate infrastructure.

Features

- It maintains high availability.
- It can manage multiple cloud environments.
- It uses popular Ruby language to create a domain-specific language.
- The chef does not make any assumptions about the current status of the node. It uses its mechanism to get the current state of the machine.

6) Jenkins

Jenkins is a DevOps tool for monitoring the execution of repeated tasks. Jenkins is a software that allows continuous integration. Jenkins will be installed on a server where the central build will take place. It helps to integrate project changes more efficiently by finding the issues quickly.

Features

- Jenkins increases the scale of automation.
- It can easily set up and configure via a web interface.
- It can distribute the tasks across multiple machines, thereby increasing concurrency.
- It supports continuous integration and continuous delivery.
- It offers 400 plugins to support the building and testing any project virtually.
- It requires little maintenance and has a built-in GUI tool for easy updates.

7) Git

Git is an open-source distributed version control system that is freely available for everyone. It is designed to handle minor to major projects with speed and efficiency. It is developed to co-ordinate the work among programmers. The version control allows you to track and work together with your team members at the same workspace. It is used as a critical distributed version-control for the DevOps tool.

Features

- It is a free open source tool.
- It allows distributed development.
- It supports the pull request.
- It enables a faster release cycle.
- Git is very scalable

CLOUD PLATFORM

There are tons of ways in which every individual can state the meaning of the cloud platform. But in the simplest way it can be stated as the operating system and hardware of a server in an Internet-based data centre are referred to as a cloud platform. It enables remote and large-scale coexistence of software and hardware goods.

Compute facilities, such as servers, databases, storage, analytics, networking, applications, and intelligence, are rented by businesses. As a result, businesses do not need to invest in data centers or computing facilities. They actually pay for the services they offer.

Types of Cloud Platforms

Cloud systems come in a range of shapes and sizes. None of them are suitable for all. To meet the varying needs of consumers, a range of models, forms, and services are available. They are as follows:

- Public Cloud: Third-party providers that distribute computing services over the Internet are known as public cloud platforms. A few good examples of trending and mostly used cloud platform are Google

Cloud Platform, AWS (Amazon Web Services), Microsoft Azure, Alibaba and IBM Bluemix.

- Private Cloud: A private cloud is normally hosted by a third-party service provider or in an on-site data centre. A private cloud platform is always dedicated to a single company and it is the key difference between the public and private cloud.
  Or we can say that a private cloud is a series of cloud computing services used primarily by one corporation or organization.
- Hybrid Cloud: The type of cloud architecture that combines both the public and private cloud systems is termed to as a Hybrid cloud platform. Data and programs are easily migrated from one to the other.

This allows the company to be more flexible while still improving infrastructure, security, and enforcement.

Organizations can use a cloud platform to develop cloud-native software, test and build them, and store, back up, and recover data. The major role of it is that will not only help the company to grow but also it helps to perform the data analysis with the help of different algorithms and the results can be a true deal breaker. Streaming video and audio, embedding information into activities, and providing applications on-demand on a global scale are all possibilities.

Benefits of cloud computing

Cloud computing represents a significant departure from how companies have traditionally seen IT services. The following are seven of the most popular reasons why businesses are moving to cloud computing services:

Cost

Cloud storage reduces the upfront costs of purchasing hardware and software, as well as the costs of setting up and operating on-site datacenters-server racks, round-the-clock power and cooling, and IT professionals to manage the infrastructure. It quickly adds up.

Global scale

The ability to scale elastically is one of the advantages of cloud computing services. In other words it simply means that we can decide the processing speed, location of the data centre where data is to be stored, storage and even the bandwidth for our process and data.

Performance

The most popular cloud computing services are hosted on a global network of protected datacenters that are updated on a regular basis with the latest generation of fast and powerful computing hardware.

Security

Many cloud providers have a comprehensive collection of policies, technologies, and controls to help us to enhance our overall security posture and protect our data, applications, and infrastructure from threats.

Speed

It means that the huge amount of calculation and the huge data retrieval as in download and upload can happen just within the blink of an eye, obviously depending on the configuration.

Reliability

Since data can be replicated at several redundant locations on the cloud provider's network, cloud storage makes data backup, disaster recovery, and business continuity simpler and less costly.

TYPES CLOUD SERVICES

IaaS, PaaS, serverless and SaaS.

Understanding what these services are and how they can make our process smooth and a lot easier by knowing their right objective will help our organisation grow more.

- **Type 1- The Infrastructure as a service (IaaS)**

Cloud computing services in their most basic form. We rent IT infrastructure-servers and virtual machines (VMs), storage, networks, and operating systems-on a pay-as-you-go basis from a cloud provider with IaaS.

- **Type 2- The Platform as a service (PaaS)**

Cloud computing platforms that provide an on-demand environment for designing, testing, distributing, and managing software applications are referred to as platform as a service. PaaS was built to make it easier for developers to build web or mobile apps easily without having to worry about setting up or maintaining the underlying infrastructure of servers, storage, network, and databases.

- **Type 3- The Software as a service (SaaS)**

Software as a service (SaaS) is a method of distributing software applications over the Internet on demand and generally by subscription. SaaS allows cloud providers to host and maintain software applications and underlying infrastructure, as well as handle maintenance such as software updates and security patching.

USES OF CLOUD COMPUTING

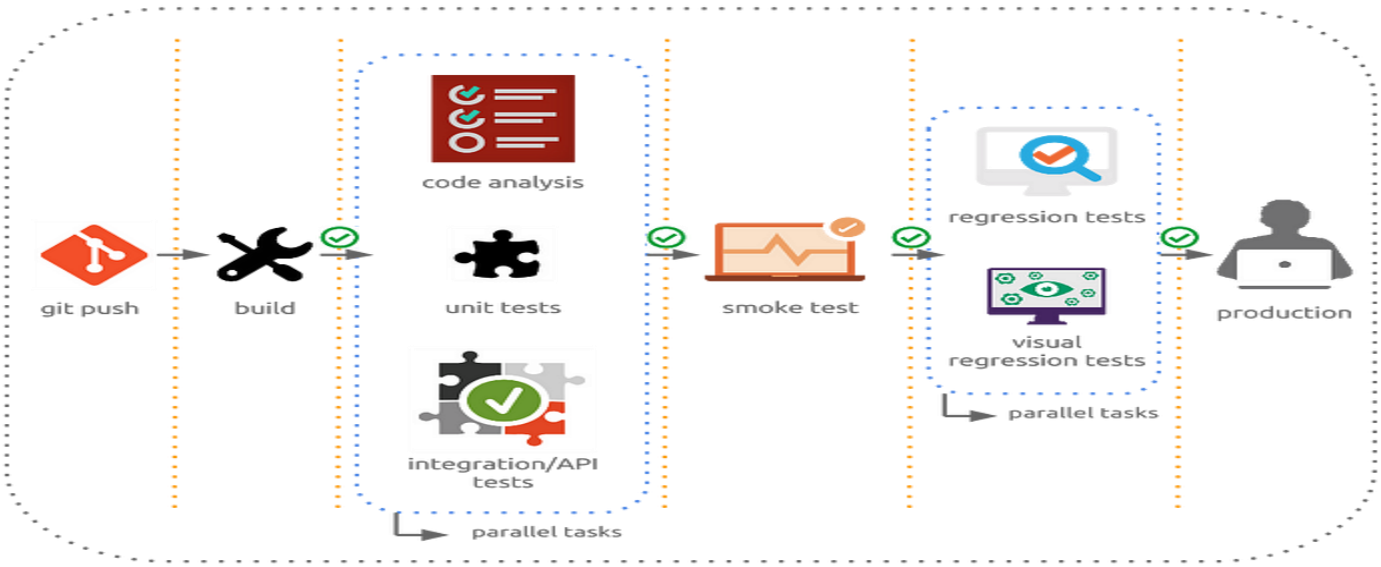
If we are watching a movie online, playing games, listening to music, sending email, purchasing things online, transferring money online almost anything we are somewhere using the cloud platform. The cloud has become the heart of the internet and somehow each and everything we use on internet is hosted on cloud.

DEPLOYMENT PIPELINE

Deployment pipeline is a concept for avoiding waste in the software development process, and it is used for providing quick feedback to the team during deployment. It works in the following way: the software deployment is divided in different stages, where tasks are run in each of these stages. In the same stage, tasks can be executed in parallel, for helping on the feedback loop. When all tasks in a stage passes, the tasks in the next stages can start

In the below image I tried to visually show you an example of a deployment pipeline, where after a developer pushes his/her code to a remote repository, the deployment pipeline starts. First building the application, then

running code analysis, unit tests, and integration/API tests (all in parallel). If all the tasks in this stage of the pipeline passes, then a smoke test suite is triggered, and if the smoke test also passes, it triggers the regression test suite and the visual regression test suite (executed in parallel), and then, if this final stage passes as well, then we have a release candidate that can be promoted to production, so that users can enjoy.



Deployment pipeline example — designed by Walmyr FilhoThe importance of using this approach, as already mentioned, is about avoiding waste. This means that if, for example, one or more tasks in some of the stages fail, it automatically fails all the deployment, and no time is waste running the other tasks unnecessarily.

Another important point about using deployment pipelines is that we can separate tasks that take longer time to execute from tasks that run faster. An example of this is a comparison of the time spent on executing a suite of unit tests versus a suite of end-to-end regression tests.deployment pipeline can be used to show the value stream map of all the deployment workflow. This can be a powerful and valuable artifact for making non-technical people understand what is necessary to deliver high-quality

software, and then having their help on providing the necessary resources to make it happens.