EnggTree.com

UNIT I

SOFTWARE PROCESS AND AGILE DEVELOPMENT

Introduction to Software Engineering, Software Process, Perspective and Specialized Process Models –Introduction to Agility-Agile process-Extreme programming-XP Process-Case Study

1. INTRODUCTION TO SOFTWARE ENGINEERING

1.1 SOFTWARE ENGINEERING

The Evolving Role of Software:

Software can be considered in a dual role. It is a product and, a vehicle for delivering a product. As a product, it delivers the computing potential in material form of computer hardware.

Example

www.EnggTree.com

A network of computers accessible by local hardware, whether it resides with in a cellular phone or operates in side a main frame computer.

i) As the vehicle, used to deliver the product. Software delivers the most important product of our time-Information.

ii) Software transforms personal data, it manages business information to enhance competiveness, it provides a gateway to worldwide information networks (e.g., Internet)and provides the means for acquiring information in all of its forms.

iii) Software acts as the basis for operating systems, networks, software tools and environments.

Software Characteristics

Software is a logical rather than a physical system element. Therefore, software hascharacteristicsthatareconsiderablydifferentthanthoseofhardware:

1. Software is developed or engineered; it is not manufactured in the classical sense.

- Although some similarities exist between the envelopment and hardware manufacture, the two activities are fundamentally different.

Downloaded from EnggTree.com

EnggTree.com

 In both activities, high quality is achieved through good design, but the manufacturing phase for hard ware can introduce quality problems that are nonexistent (or easily corrected) for software. Software doesn't "wear out."

1.1.2. Software Application Domains

The following categories of computer software present continuing challenges for software engineers.

a) System software:

- System software is a collection of programs written to service of their programs.
- Example: compilers, editors, and file management utilities, operating system components, drivers, telecommunications processors, process largely indeterminate data.

b) Real-time software:

Elements of real-time software includes

- a data gathering component that collects and formats information from an
- external environment an analysis component that transforms information as
- required by the application
- a control/output component that responds to the external environment

a monitoring component that coordinates all other components so that real-timeresponse(typicallyrangingfrom1millisecondto1second)canbemaintained.

www.EnggTree.com

c) Business software:

- Business information processing is the largest single software application
- area.

Example: payroll, accounts receivable/payable, inventory.

Applications in this area restructure existing data in a way that facilitates business operations or management decision making. In addition to conventional data processing application, business software applications also encompass interactive computing

- Example: pointof-sale transaction processing.

d) Engineering and scientific software:

- This is the software using "number crunching"
- algorithms.

Example: System simulation,computer-aideddesign.

e) Embedded software:

Downloaded from EnggTree.com

EnggTree.com

- Embedded software resides inread-only memory and is used to control products and systems for the consumer and industrial markets.

SOFTWAREENGINEERING

In order to build software that is ready to meet the challenges and it must recognize a few simple realities:

- It follows that a concerted effort should be made to understand the problem before as of ware solution is developed.
- It follows that design becomes a pivotal activity.
- It follows that software should exhibit high quality. It follows that software
- should be maintainable.

These simple realities lead to one conclusion: software in all of its forms and a cross allow fits application domains should be engineered.

- Software engineering is the establishment and use of sound engineering principles in order to obtain economically software that is reliable and works efficiently on real machines.
- Software engineering encompasses a process, methods form an aging and engineering software, and tools.

www.EnggTree.com

THE SOFTWARE PROCESS

- A **process** is a collection of activities, actions, and tasks that are performed when some work product is to be created.
- An **activity** strives to achieve a broad objective and is applied regardless of the application domain, size of the project, complexity of the effort, or degree of rigor with which software engineering is to be applied.
- An **action** (e.g., architectural design) encompasses a set of tasks that produce a major work product(e.g.,anarchitectural design model).
- A **task** focuses on a small, but well-defined objective(e.g., conducting a unit test)that produces tangible outcome.

✓ A process framework establishes the foundation for a complete software engineering process by identifying a small number of framework activities that are applicable to all software projects, regardless of their size or complexity.

✓ In addition, the process framework encompasses a set of **umbrella activities** that are applicable across the entire software process. A generic process frame work for software engineering encompasses five activities:

Downloaded from EnggTree.com

EnggTree.com

The five generic process frame work activities:

a) Communication:

- The intent is to understand stakeholders' objectives for the project and to gather requirements that help define software features and functions.

b) Planning:

- Software project plan—defines the software engineering work by describing the technical tasks to be conducted, the risks that are likely, the resources that will be required, the work products to be produced, and a work schedule.

c) Modeling:

- A software engineer does by creating models to better understand software requirements and the design that will achieve those requirements.

d) Construction:

- This activity combines code generation (either manual or automated) and the testing that is required to un cover errors in the code.

e) Deployment:

www.EnggTree.com

Downloaded from EnggTree.com

EnggTree.com

- Software engineering process framework activities are complemented by a number of umbrella activity.
- In general, umbrella activities are applied throughout a software project and help a software teammanageandcontrolprogress, quality,change, andrisk.Typicalumbrellaactivitiesinclude:

i) **Software project tracking and control**—allowsthesoftwareteamtoassessprogressagainsttheprojectplanandtake any necessary action to maintain the schedule.

ii) **Risk management**—assesses risks that may affect the out come of the project or the quality of the product.

iii) **Software quality assurance**—defines and conducts the activities required to ensure software quality.

iv) **Technical reviews**—access software engineering work products in an effort to uncover and remove errors before they are propagated to the next activity.

v) **Measurement**—defines and collects process, project, and product measures that assist the teaming delivering software that meets stakeholders' needs; can be used in conjunction with all other frame work and umbrella activities.

vi) **Software configuration management**—manages the effects of change throughout the software process.

vii) **Reusability management**—defines criteria for work product reuse(includingsoftwarecomponents)andestablishesmechanismstoachieve reusable components.

viii) **Workproductpreparationandproduction**—encompassestheactivitiesrequiredtocreateworkproductssuchasmodels, documents, logs, forms, and lists.

www.EnggTree.com

Downloaded from EnggTree.com

EnggTree.com

3. PRESCRIPTIVE PROCESS MODELS

- ✓ Prescriptive process models were originally proposed to bring order to the software development.
- ✓ Prescriptive process models define a prescribed set of process elements and a predictable process workflow.

▪ Prescriptive Process Models

- The Waterfall Model
- Incremental Process Models Evolutionary Process Modes

1.1TheWaterfallModel

The waterfall model, sometimes called the classic lifecycle, suggests asystematic,sequential approach to software development that begins with customer specification of requirements and progresses through planning,modelling, construction, and deployment, culminating in ongoing support of the completed software.

- A variation in the representation of the water fall model is called the **V-model.**
- Represented in Figure 1.5, the V-model [Buc99] depicts the relationship of quality assurance actions to the actions associated with communication, modeling, and early construction activities.

www.EnggTree.com

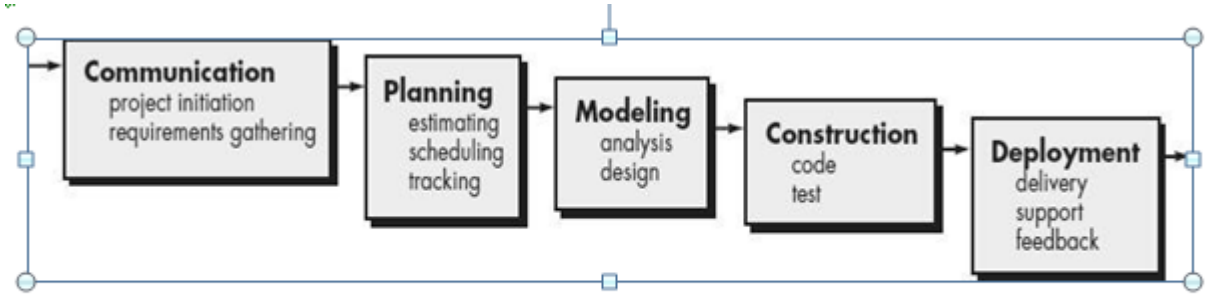


FIGURE1.5Thewaterfallmodel

1.2. Incremental Process Models:

The incremental model deliver saseries of releases,called increments that provide

Downloaded from EnggTree.com

EnggTree.com

progressively more functionality for the customer as each increment is delivered.

- The incremental model applies linear sequences in as staggered fashion as calendar time progresses.
- Each line are sequence produces deliverable"increments"of the software [McD93]in a manner that is similar to the increments produced by an evolutionary process flow.
- The first increment is called core product. In this release the basic requirements areimplementedandtheninsubsequentincrementsnewrequirementsareadded.
- The core product is used by the customer (or under goes detailed evaluation).
- As a result to fuse and/or evaluation, a plan is developed for then increment. The plan addresses the modification of the core product to better meet the needs of the customer and the delivery of additional features and functionality. This process is repeated following the delivery of each increment, until the complete product is produced.

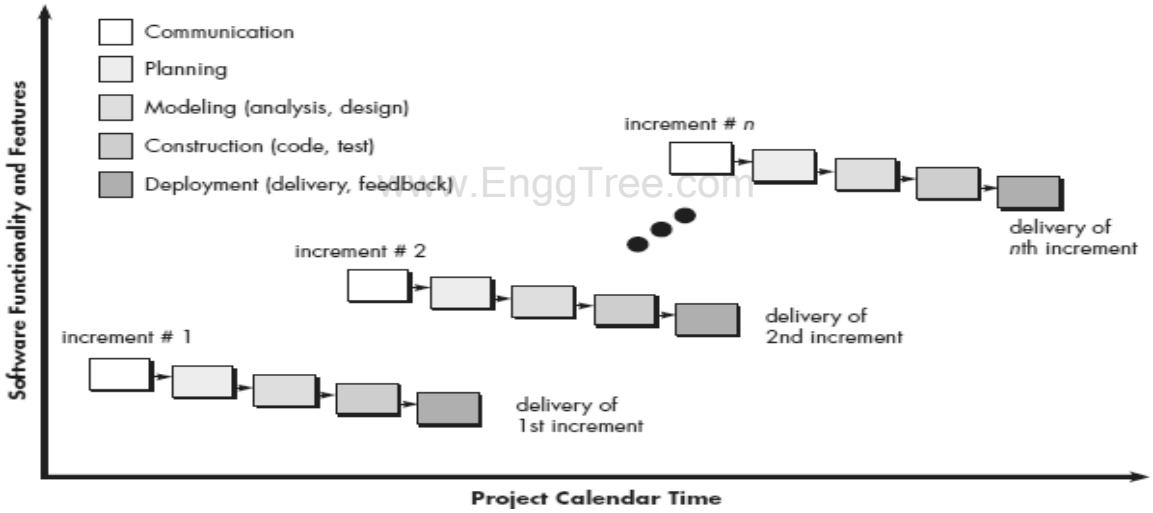


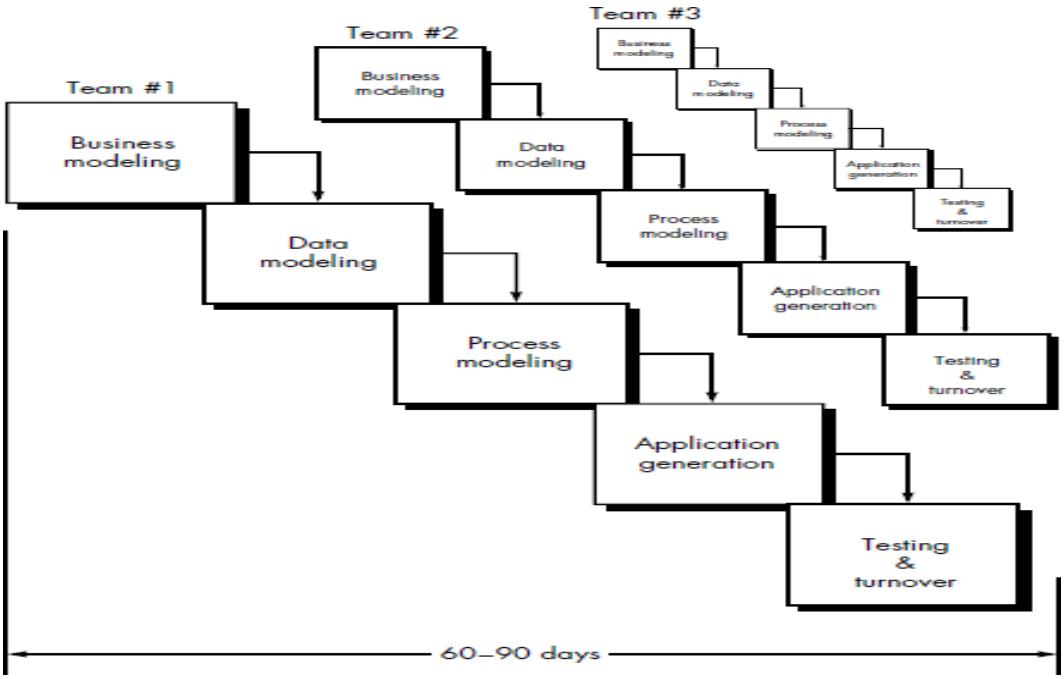
Figure1.7 IncrementalProcessModel

3.TheRAD Model

- Rapid Application Development is a line sequential software development process model that emphasizes an extremely short development cycle.
- Rapid application achieved by using a component based construction approach.
- If requirements are well understood and projects copies constrained the RAD process enables a development team to create a—f ully functional system.

Downloaded from EnggTree.com

EnggTree.com



www.EnggTree.com

1.1.1.1 The Prototyping Model:

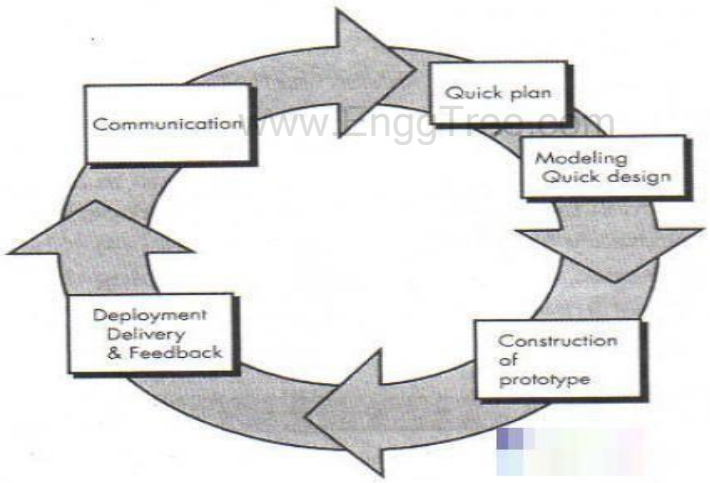
- The proto typing paradigm begins with communication. Developer and customer meet and define the overall objectives for the software, identify whatever requirements are known,

Downloaded from EnggTree.com

EnggTree.com

- A **quick design** focuses on a representation of those aspects of the software that will be visible to the customer/user (e.g. Input approaches and output formats). The quick design leads to the construction of a prototype.
- The prototype is evaluated by the customer/user and used to refine requirements for the software to be developed. Iteration occurs as the prototype is tuned to satisfy the needs of the customer, while at the same time enabling the developer to better understand what needs to be done.
- Ideally ,the proto type serves as a mechanism for identifying software
- requirements.

  If a working proto type is built, the developer attempts to use existing program fragments or applies tools (e.g., report generators, window managers) that enable working programs to be generated quickly.



4.SpiralModel:

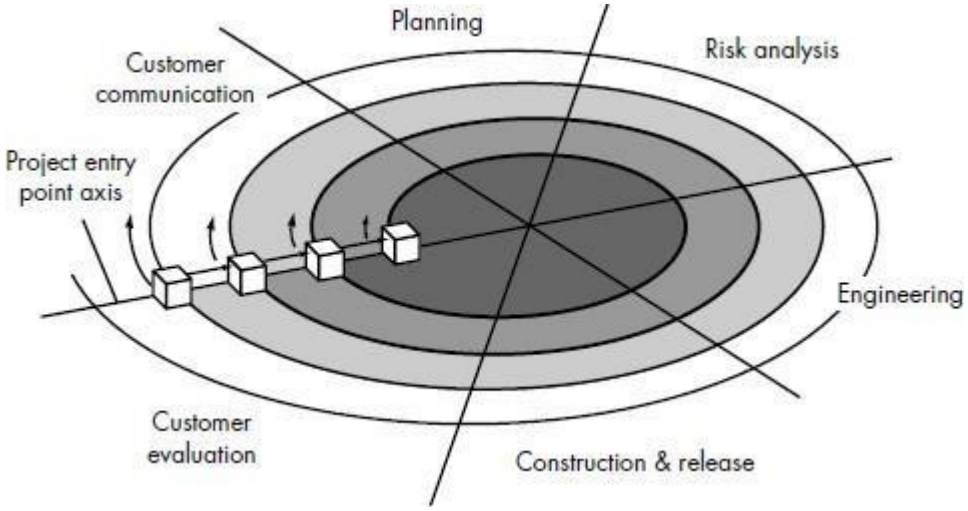
- The spiral model is an evolutionary software process model that couples the iterative nature of prototyping with the control led and systematic aspects of the linear sequential model.
- The spiral development model is a risk-driven process model generator that is used to guide multi-stake holder concurrent engineering of software intensive systems.
- That domain distinguishing features.
- One is a cyclic approach for incrementally growing a system's degree of definition

Downloaded from EnggTree.com

EnggTree.com

and implementation while decreasing its degree of risk.

- The other is a set of anchor point milestones for ensuring stakeholder commitment to feasible and mutually satisfactory system solutions.
- Using the spiral model, software is developd in a series of incremental releases.
- A spiral model is divided into a number of framework activities, also called **task**
- **regions**



www.EnggTree.com

Downloaded from EnggTree.com

EnggTree.com

INTRODUCTION TO AGILITY:

- Agility is the ability to **respond quickly to changing needs.** It encourage steam structures and attitude sthat make **effective communication among all stakeholders.**
- It emphasizes **rapid delivery of operational of tware** and deemphasizes the importance of intermediate work products.
- It adopts the **customer as a part of the development team**.
- It helps in **organizing a teams ot habits in control of the work performed.Yielding**
- Agility results in **rapid, incremental delivery of software.**

Agility and the Cost of Change:

- The cost of change in software development increases nonlinearly as a project progresses (Figure1.13, solidblackcurve).
- It is relatively easy to accommodate a change when software team gathered its requirements.
- The costs of doing this work are minimal, and the time required will not affect the outcome of the project.

www.EnggTree.com

- Cost varies quickly, and the cost and time required ensuring that the change is made without any side effects is non trivial.
- **An agile process reduces the cost of change because software is released in increments and changes can be better controlled with in an increment.**
- Agile process “flattens” the cost of change curve (Figure 1.11, shaded, solid curve), allowing a software team to accommodate changes late in a software project without dramatic cost and time impact.
- When incremental delivery is coupled with other agile practices such as continuous unit testing and pair programming, the cost of making a change is attenuated.

Downloaded from EnggTree.com

EnggTree.com

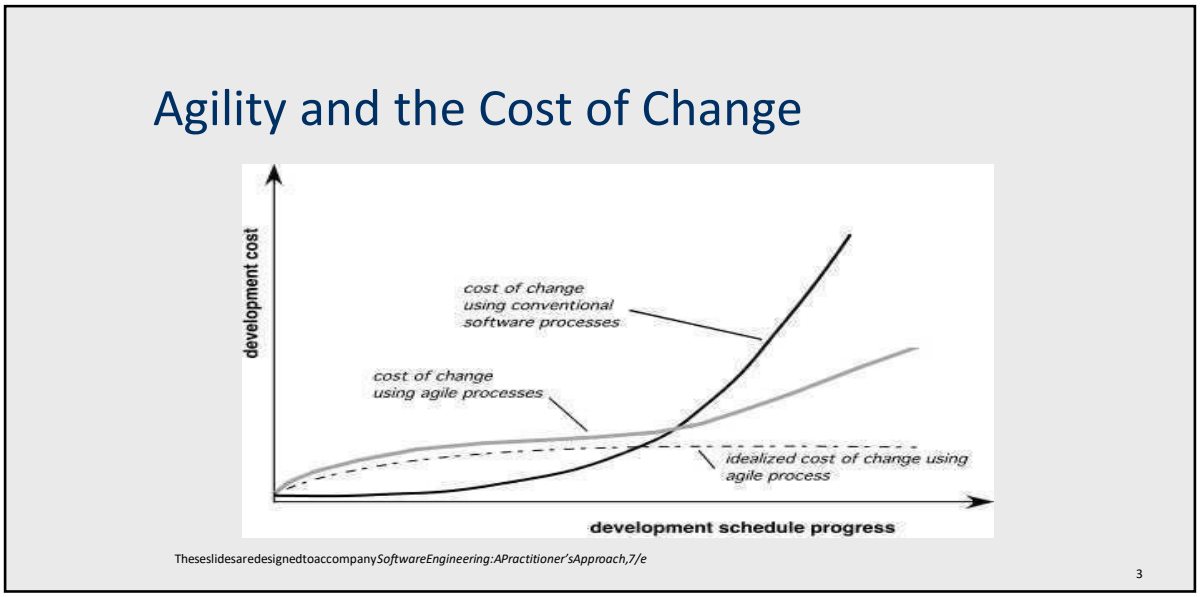


Figure1.13Change costs as a function of time in development

4. AN AGILE PROCESS

An Agile Process is characterized in a manner that addresses a number of key assumptions about the majority of software project:

1. It is difficult to predict which software requirements will persist and which will change.
2. It is difficult to predict those customer priorities will change.
3. It is difficult to predict them much design is necessary before construction.
4. Analysis, design, construction, and testing are not as predictable.

AGILITY PRINCIPLES:

1. To **satisfy the customer through early and continuous delivery of software**.
2. **Welcome changing requirements**, even late in development.
3. **Deliver working software frequently**, from a couple of weeks to a couple of months.
4. **'Customers and developers must work together daily** throughout the project.
5. **Build projects around motivated individuals**.
6. **Emphasis on face-to-face communication.**
7. **Workings of tware are the primary measure of progress**.
8. Agile processes promote **sustainable development**.
9. Continuous attention to **technical excellence and good design enhances agility**.

Downloaded from EnggTree.com

EnggTree.com

10. **Simplicity**—the art of maximizing the amount of work not done—**is essential**.
11. **Self-organizing teams produce the best architectures/requirements/design**.
12. **The team reflects on how to become more effective** at regular intervals.

13. Human Factors:

- Agile development focuses on **the talents and skills of individuals, molding the process to specific people and teams.**
- ***The process molds to the needs of the people and team***,not the other way around.

A number of key traits must exist among the people on an agile team and the team itself:

- ✓ Competence.
- ✓ Common focus.
- ✓ Collaboration.
- ✓ Decision-making ability.
- ✓ Fuzzy problem-solving ability.
- ✓ Mutual trust and respect.
- ✓ Self-organization**.**

www.EnggTree.com

Downloaded from EnggTree.com

EnggTree.com

7. **The XP Process:**

 Extreme programming uses an **object-oriented approach** for software development. There are four frame work activities involved in XP Process.

1. Planning
2. Designing
3. Coding
4. Testing

1. Planning:

- Begins with the creation of a set of stories (also called use stories).Each story is written by the customer and is placed on an in dexcard.The customer assigns a value(i.e. apriority)to the story.
- Agile team assesses each story and assigns a cost. Stories are grouped to for a deliverable increment.

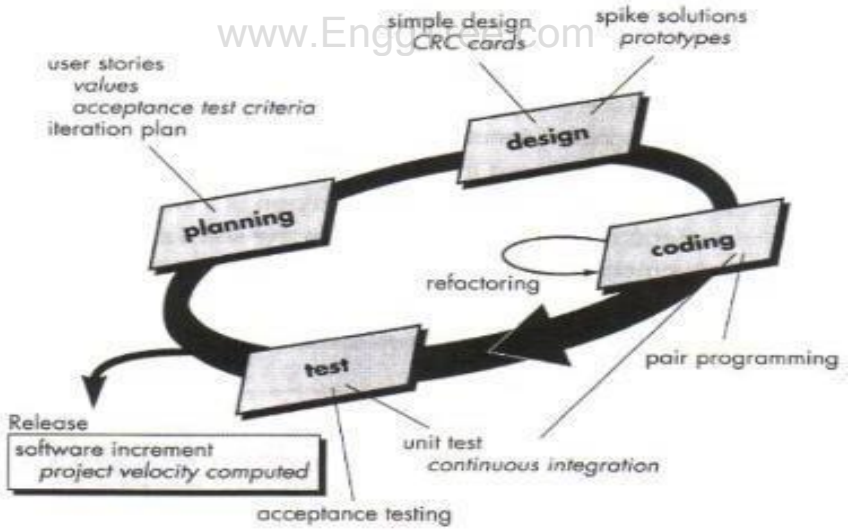


Figure1.15The Extreme Programming Process

- A commitment is made on delivery date.
- After the first increment "project velocity "is used to help define sub sequent delivery dates for other increments.

2. Design:

Downloaded from EnggTree.com

EnggTree.com

- Follows the keep it simple principle.
- Encourage the use of CRC(class-responsibility-collaborator)cards.
- For difficult design problems, suggests the creation of "spike solutions"—a design prototype. Encourages"refactoring"—an iterative refinement of the internal program design

Design occurs both before and after coding commences.

3. Coding:

- Recommends the construction of a series of unit tests for each of the stories before coding

Test driven management establishes a series of measurable"destinations"and then defines mechanisms for determining whether or not these destinations have been reached.

Retrospectives:

i) An IXP team conducts at view after software increment is delivered called retrospective.

ii) This review examines "issues, events, and lessons-learned"acrossasoftware increment and/or the entire software release.

www.EnggTree.com

iii) The intent is to improve the IXP process.

Continuous learning:

i) Learning is a vital product of continuous process improvement; members of the XP team are encouraged to learn new methods and techniques that can lead to a higher quality product.

ii) In addition to theses ix new practices, IXP modifies a number of existing XP practices.

✓ **Storyrivendevelopment(SDD):**

✓ Insists that stories for acceptance tests be written be for easing line of code is developed.

EXTREME PROGRAMMING (XP):

- The best known and a very influential agile method, Extreme Programming(XP)takes an'extreme'approach to iterative development.

✓ New versions may be built several times per day;

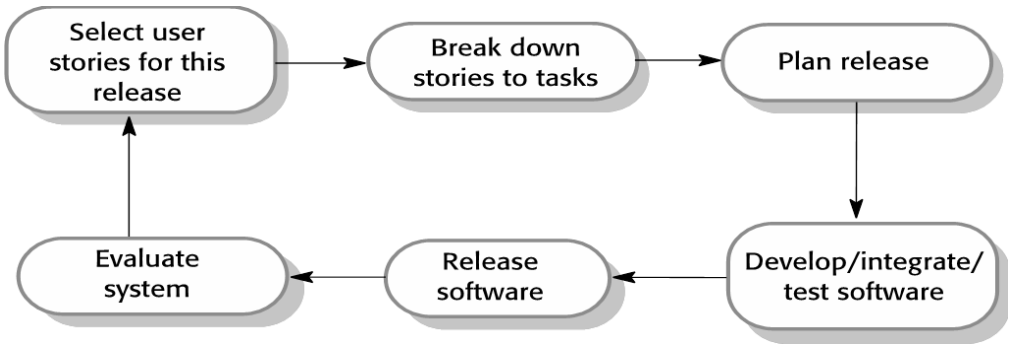
✓ Incrementsaredeliveredtocustomersevery2weeks;

✓ Alltestsmustberunforeverybuildandthebuildisonlyacceptediftestsrunsuccessf ully.

Downloaded from EnggTree.com

EnggTree.com

- ✓ This is how XP supports **agile principles**:



- ✓

Figure1.14The extremeprogramming release cycle

- People not process through **pair programming, collective ownership** and a process that avoids long working hours.
- Change supported through **regular system releases**. Maintaining simplicity through **constant refactoring of code.**

6.1XP values:

- ✓ XP is comprised of five values such as:

i. Communication

www.EnggTree.com

ii. Simplicity
iii. Feedback
iv. Courage
v. Respect.

- ✓ Each of these values is used as a driver for specific X P activities, actions, and task.
- ✓ In order to achieve effective **communication** between **software engineers and other stake holders**, XP emphasizes close, yet informal(verbal) collaboration between customers and developers, the establishment of effective metaphors for communicating important concepts, continuous feedback, and the avoidance of volume in documentation as a communication medium.
- ✓ To consider **simplicity**, XP restricts developers to design only for immediate needs, rather than future needs.
- ✓ **Feedback** is derived from three sources: the **software, the customer and other team members**.
- ✓ By designing and implementing an effective testing strategy, the software provides the agile team with feedback.
- ✓ The team develops a **unit test** for each class being developed, to exercise each

Downloaded from EnggTree.com

EnggTree.com

operation according to its specified functionality.

- ✓ The **user stories or use cases** are implemented by the increments being used as a basis for acceptance tests. The degree to which software implements the **output, function, and behavior of the test case** is a form of feedback.
- ✓ An agile XP team must have the courage (discipline) to design for today, recognizing that future requirements may change dramatically, thereby demanding substantial rework of the design and implemented code.

www.EnggTree.com

Downloaded from EnggTree.com

EnggTree.com

Unit -II

SOFTWARE REQUIREMENT SPECIFICATION

Requirement analysis and specification – Requirements gathering and analysis – Software Requirement Specification – Formal system specification – Finite State Machines – Petri nets – Object modelling using UML – Use case Model–Class diagrams – Interaction diagrams–Activity diagrams–State chart diagrams –Functional modeling–Data Flow Diagram.

---

1. Requirement analysis and specification

The **Requirement Analysis** and **Specification** phase starts after the feasibility study stage is complete and the project is financially viable and technically feasible. This phase ends when the requirements specification document has been developed and reviewed. It is usually called the **Software Requirements Specification (SRS)** Document.

ww.EnggTree.com

These activities are usually carried out by a few experienced members of the development team and it normally requires them to spend some time at the customer site. The engineers who gather and analyse customer requirements and then write the requirements specification document are known as **system analysts** in the software industry. System analysts collect data about the product to be developed and analyse the collected data to conceptualize what exactly needs to be done. After understanding the precise user requirements, the analysts analyse the requirements to weed out inconsistencies, anomalies and incompleteness.

We can conceptually divide the requirements gathering and analysis activity into two separate tasks:

- Requirements gathering
- Requirements analysis

REQUIREMENTS GATHERING:

Downloaded from EnggTree.com

EnggTree.com

It is also known as **Requirements Elicitation**. The primary objective of the requirements- gathering task is to **collect the requirements from the stakeholders**. A **stakeholder** is a source of the requirements and is usually a person or a group of persons who either directly or indirectly are concerned with the software.

1. Studying existing documentation:

The analyst usually studies all the available documents regarding the system to be developed before visiting the customer site. Customers usually provide a statement of purpose (SoP) document to the developers. Typically, these documents might discuss issues such as the context in which the software is required.

2. Interview:

Typically, there are many different categories of users of the software. Each category of users typically requires a different set of features from the software. Therefore, it is important for the analyst to first identify the different categories of users and then determine the requirements of each.

3. Task analysis:

The users usually have a black-box view of software and consider the software as something that provides a set of services. A service supported by the software is also called a **task**. We can therefore say that the software performs various tasks for the users. In this context, the analyst tries to identify and understand the different tasks to be performed by the software. For each identified task, the analyst tries to formulate the different steps necessary to realize the required functionality in consultation with the users.

4. Scenario analysis:

A task can have many scenarios of operation. The different scenarios of a task may take place when the task is invoked under different situations. For different types of scenarios of a task, the behaviour of the software can be different.

5. Form analysis:

Form analysis is an important and effective requirement-gathering activity that is undertaken by the analyst when the project involves automating an

Downloaded from EnggTree.com

EnggTree.com

existing manual system. During the operation of a manual system, normally several forms are required to be filled up by the stakeholders, and in turn, they receive several notifications.

In form analysis, the existing forms and the formats of the notifications produced are analyzed to determine the data input to the system and the data that are output from the system.

REQUIREMENT ANALYSIS AND SPECIFICATION:

After requirements gathering is complete, the analyst analyses the gathered requirements to form a clear understanding of the exact customer requirements and to weed out any problemsin the gathered requirements. It is natural to expect the data collected from various stakeholders to contain several contradictions, ambiguities, and incompleteness.

The main purpose of the requirements analysis activity is to analyse the gathered requirements to remove all ambiguities, incompleteness, and inconsistencies from the gathered customer requirements and to obtain a clear understanding of the software to bedeveloped.

During requirements analysis, the analyst needs to identify and resolve three main types of problems in the requirements

CHARACTERISTICS OF GOOD SRS

Downloaded from EnggTree.com

EnggTree.com

Following are the features of a good SRS document:

1. **Correctness:** User review is used to provide the accuracy of requirements stated in the SRS. SRS is said to be perfect if it covers all the needs that are truly expected from the system.

2. **Completeness:** The SRS is complete if, and only if, it includes the following elements:

(1). All essential requirements, whether relating to functionality, performance, design, constraints, attributes, or external interfaces.

(2). Definition of their responses of the software to all realizable classes of input data in all available categories of situations.

(3). Full labels and references to all figures, tables, and diagrams in the SRS and definitions of all terms and units of measure.

www.EnggTree.com

3. **Consistency:** The SRS is consistent if, and only if, no subset of individual requirements described in its conflict. There are three types of possible conflict in the SRS:

(1). The specified characteristics of real-world objects may conflicts. For example, one requirement as tabular but in another as textual.

Unambiguousness: SRS is unambiguous when every fixed requirement has only one interpretation. This suggests that each element is uniquely interpreted. In case there is a method used with multiple definitions, the requirements report should determine the implications in the SRS so that it is clear and simple to understand.

Ranking for importance and stability: The SRS is ranked for importance and stability if each requirement in it has an identifier to indicate either the significance or stability of that particular requirement.

Typically, all requirements are not equally important. Some prerequisites may be essential, especially for life-critical applications, while others may be desirable.

Downloaded from EnggTree.com

EnggTree.com

Each element should be identified to make these differences clear and explicit. Another way to rank requirements is to distinguish classes of items as essential, conditional, and optional.

Modifiability: SRS should be made as modifiable as likely and should be capable of quicklyobtain changes to the system to some extent. Modifications should be perfectly indexed and cross-referenced.

Verifiability: SRS is correct when the specified requirements can be verified with a cost- effective system to check whether the final software meets those requirements. The requirements are verified with the help of reviews.

Traceability: The SRS is traceable if the origin of each of the requirements is clear and if it facilitates the referencing of each condition in future development or enhancementdocumentation.

www.EnggTree.com

Design Independence: There should be an option to select from multiple design alternatives for the final system. More specifically, the SRS should not contain any implementation details.

Testability: An SRS should be written in such a method that it is simple to generate test cases and test plans from the report.

Understandable by the customer: An end user may be an expert in his/her explicit domain but might not be trained in computer science. Hence, the purpose of formal notations and symbols should be avoided too as much extent as possible. The language should be kept simple and clear.

The right level of abstraction: If the SRS is written for the requirements stage, the details should be explained explicitly. Whereas,for a feasibility study, fewer analysis can be used. Hence, the level of abstraction modifies according to the objective of the SRS.

The format of an output report may be described in

Downloaded from EnggTree.com

EnggTree.com

- Anomaly
- Inconsistency
- Incompleteness

Anomaly: It is an anomaly is an ambiguity in a requirement. When a requirement is anomalous, several interpretations of that requirement are possible. Any anomaly in any of therequirements can lead to the development of an incorrect system, since an anomalous requirement can be interpreted in several ways during development.

Inconsistency: Two requirements are said to be inconsistent if one of the requirementscontradicts the other.

Incompleteness: An incomplete set of requirements is one in which some requirements have been overlooked. The lack of these features would be felt by the customer much later, possibly while using the software. Often, incompleteness is caused by the inability of the customer to visualise the system that is to be developed and to anticipate all the features that would be required

www.EnggTree.com

Downloaded from EnggTree.com

EnggTree.com

www.EnggTree.com

Downloaded from EnggTree.com

EnggTree.com

2. FORMAL SYSTEM SPECIFICATION

Formal system specification in software engineering refers to the process of precisely describing the behavior, properties, and requirements of a software system using formal methods and mathematical notations. It aims to eliminate ambiguity, clarify system requirements, and enable formal analysis and verification of system properties.

Here are some key aspects of formal system specification:

Formal Methods:

Formal methods are mathematical techniques used to specify, model, and reason about software systems. They provide a rigorous and precise approach to system specification, ensuring clarity and unambiguous representation. Common formal methods used for system specification include Z notation, B method, and Specification and Description Language (SDL).

www.EnggTree.com

Mathematical Notations:

Formal system specification often involves using mathematical notations to express system requirements, constraints, and behaviors. These notations can include predicate logic, set theory, algebraic expressions, temporal logic, and process calculi, dependingon the chosen formal method.

Language Constructs:

Formal specification languages provide constructs and syntax for expressing system properties. For example, Z notation includes constructs for defining data types, predicates, schema definitions, and invariants. These language constructs allow for a systematic and structured representation of the system.

Requirements Capture:

Formal system specification helps in capturing and refining system requirements. It enables the identification of inconsistencies, ambiguities, and gaps in

Downloaded from EnggTree.com

EnggTree.com

Requirements early in the development process. Formal methods facilitate the translation of informal requirements into precise and unambiguous specifications.

Verification and Validation:

Formal system specification enables formal analysis, verification, and validation of system properties. Formal methods support the use of automated tools and theorem provers to check the consistency, correctness, and completeness of system specifications. This helps in identifying design flaws, detecting logical inconsistencies, and verifying that the system meets its intended requirements.

Refinement:

Formal specification allows for stepwise refinement of system specifications. System requirements can be progressively refined into more detailed specifications, ensuring that the design and implementation faithfully adhere to the original requirements.

www.EnggTree.com

Documentation:

Formal system specifications serve as a valuable source of documentation. They provide a clear and unambiguous representation of system requirements, constraints, and behavior, which aids in system understanding, maintenance, and future enhancements.

Formal system specification plays a crucial role in software engineering by enhancing the quality, reliability, and maintainability of software systems. It supports rigorous analysis, verification, and validation, leading to more dependable and trustworthy software designs.

3. FINITE STATE MACHINE

- A state machine is a software computation model. It's just a model to solve a complex application, and it comprises a finite number of states. Hence it is also called a Finite State Machine. States are nothing but situations of your application (different situations).
- Since states are finite, there is a finite number of transitions among the states.

Downloaded from EnggTree.com

EnggTree.com

Transitions are triggered by the incidents or input events fed to the state machine. An FSMis an event-driven reactive system.

- A state machine also produces an output. The output produced depends on the current state of the state machine sometimes, and sometimes it also depends on the input events fedto the state machine.

Benefits of using state machines:

- It is used to describe situations or scenarios of your application (Modelling the lifecycle of a reactive object through interconnections of states.

www.EnggTree.com

Downloaded from EnggTree.com

EnggTree.com

- FSMs are helpful to model complex applications that involve lots of decision-making, producing different outputs, and processing various events.
- State machines are visualized through state machine diagrams in the form of statecharts, which helps to communicate between non-developers and developers.
- FSM makes it easier to visualize and implement the changes to the behavior of theproject.
- Complex application can be visualized as a collection of different states processing afixed set of events and producing a fixed set of outputs.
- Loose coupling: An application can be divided into multiple behaviors or state machines, and each unit can be tested separately or could be reused in other applications.
- Easy debugging and easy code maintenance.
- Scalable
- Narrow down the whole application completely to state-level complexity, analyze andimplement.

www.EnggTree.com

Different types of state machines:

1. Mealy machine
2. Moore machine
3. Harel state charts
4. UML state machines

Some of these state machines are used for software engineering, and some state machines arestill being used in digital electronics, VLSI design, etc.

Types of Finite State Machines

Mealy machines are a type of simple state machine where the next state is determined by thecurrent state and the given input.

The next state is determined by checking which input generates the next state with the current state. Imagine a button that only works when you're logged in.

Downloaded from EnggTree.com

EnggTree.com

That button is a state machine with login as the current state.

The new state is determined by logging in. Once you're logged in, the next state is determined by the current state which is logged in.

Moore Machines

A Moore State Machine is a type of state machine for modeling systems with a high degree of uncertainty. In these systems, predicting the exact sequence of future events is difficult. As a result, it is difficult to determine the next event.

A Moore model captures this uncertainty by allowing the system to move from one state to another based on the outcome of a random event.

A Moore model has many applications in both industry and academia. For example, it can be used to predict when a system will fail or when certain events will occur with a high probability (e.g., when there will be an earthquake).

www.EnggTree.com

It can also be used as part of an optimization algorithm when dealing with uncertain inputs (e.g., produce only 1% more product than standard).

In addition, Moore models are often used as rules for automatic control systems (e.g., medical equipment) that need to respond quickly and accurately without human intervention.

Turing Machine

The Turing Machine consists of an input tape (with symbols on it), an internal tape (which corresponds to memory), and an output tape (which contains the result).

A Turing Machine operates through a series of steps: it scans its input tape, reads out one symbol at a time from its internal tape, and then applies this symbol as a command (or decision) to its output tape. For example: "If you see 'X' on the input tape, then print 'Y' on the output tape."

The input tape can be considered a finite set of symbols, while the internal and

Downloaded from EnggTree.com

EnggTree.com

output tapes areinfinite. The Turing Machine must read an entire symbol from its internal tape before it can move its head to the next symbol on the input tape.

Once it has moved its head to the next symbol, it can read that symbol out of its internal tape and then move to the next symbol on its input tape.

This process continues until no more input or output symbols are left in the Turing Machine's internal or external tapes (at which point, it stops).

www.EnggTree.com

Downloaded from EnggTree.com

EnggTree.com

State Machine Use Cases

State machines are helpful for a variety of purposes. They can be used to model the flow of logic within a program, represent the states of a system, or for modeling the flow of events in a business process.

There are many different types of state machines, ranging from simple to highly complex. A few common use cases include:

Modeling Business Workflow Processes

State machines are ideal for modeling business workflows. This includes account setup flows, order completion, or a hiring process.

These things have a beginning and an end and follow some sort of sequential order. State machines are also great for modeling tasks that involve conditional logic.

www.EnggTree.com

Business Decision-Making

Companies pair FSMs with their data strategy to explore the cause and effect of business scenarios to make informed business decisions.

Business scenarios are often complex and unpredictable. There are many possible outcomes, and each one impacts the business differently.

A simulation allows you to try different business scenarios and see how each plays out. You can then assess the risk and determine the best course of action.

Downloaded from EnggTree.com

EnggTree.com

www.EnggTree.com

Downloaded from EnggTree.com

EnggTree.com

4. PETRI NETS

Petri nets are a mathematical modeling tool used in software engineering to analyze and describe the behavior of concurrent systems. They were introduced by Carl Adam Petri in the 1960s and have since been widely used in various fields, including software engineering.

Petri nets provide a graphical representation of a system's state and the transitions between those states. They consist of places, transitions, and arcs. Places represent states, transitions represent events or actions, and arcs represent the flow of tokens (also called markings) between places and transitions.

In software engineering, Petri nets can be used for various purposes, including:

System Modeling: Petri nets are used to model the behavior of complex software systems, especially concurrent and distributed systems. They help in understanding and visualizing how different components of the system interact and how their states change over time.

www.EnggTree.com

Specification and Verification: Petri nets can be used to specify the desired behavior of a software system. By modeling the system using Petri nets, it becomes possible to formally verify properties such as safety, liveness, reachability, and deadlock-freeness. Verification techniques based on Petri nets help in identifying design flaws, potential deadlocks, or other issues early inthe development process.

Performance Analysis: Petri nets can be used to analyze the performance of software systems, including their throughput, response time, and resource utilization. By modeling the system's behavior and introducing timing annotations, it becomes possible to perform quantitative analysis and predict system performance under different conditions.

Workflow Modeling: Petri nets are often used to model business processes and workflows. In software engineering, this is particularly useful for modeling the flow of tasks and activities in software development processes, such as agile methodologies or continuous integration/continuousdeployment (CI/CD) pipelines.

Downloaded from EnggTree.com

EnggTree.com

Software Testing: Petri nets can be utilized in software testing to generate test cases and verify the correctness of the system. By modeling the system's behavior and different scenarios, it becomes possible to systematically generate test cases that cover various paths and states, helping inidentifying potential bugs and ensuring the software's reliability.

5. OBJECT MODELLING USING UML

UML (Unified Modeling Language) can be called a **graphical modeling language** that is used in the field of software engineering. It specifies, visualizes, builds, and documents the softwaresystem's artifacts (main elements). It is a visual language, not a programming language. UML diagrams are used to depict the behavior and structure of a system. UML facilitates modeling, design, and analysis for software developers, business owners, and system architects. A pictureis worth a thousand words.

The Benefits of Using UML

www.EnggTree.com

- Because it is a general-purpose modeling language, it can be used by any modeler. UML is a straightforward modeling approach utilized to describe all practical systems.
- Because no standard methods were available then, UML was developed to systemize and condense object-oriented programming. Complex applications demand collaboration and preparation from numerous teams, necessitating a clear and straightforward means of communication among them.
- The **UML diagrams** are designed for customers, developers, laypeople, or anyone who wants to understand a system, whether software or non-software. Businesspeople do not understand code. As a result, UML becomes vital for communicating the system's essential requirements, features, and procedures to non-programmers. Technical details became easier to understand by non-technical people through an introduction to UML.
- When teams can visualize processes, user interactions, and the system's static structure,they save a lot of time in the long run.

Characteristics of UML

Downloaded from EnggTree.com

EnggTree.com

The UML has the following characteristics:

- It is a modeling language that has been generalized for various use cases.
- It is not a programming language; instead, it is a graphical modeling language that usesdiagrams that can be understood by non-programmers as well.
- It has a close connection to object-oriented analysis and design.
- It is used to visualize the system's workflow.

Conceptual Modeling

Before moving ahead with the concept of UML, we should first understand the basics of the conceptual model.

A conceptual model is composed of several interrelated concepts. It makes it easy to understand the objects and how they interact with each other. This is the first step before drawing UML diagrams.

www.EnggTree.com

Following are some object-oriented concepts that are needed to begin with UML:

- **Object:** An object is a real world entity. There are many objects present within a singlesystem. It is a fundamental building block of UML.

Class: A class is a software blueprint for objects, which means that it defines the variables and methods common to all the objects of a particular type.

- **Abstraction:** Abstraction is the process of portraying the essential characteristics of an object to the users while hiding the irrelevant information. Basically, it is used to envision the functioning of an object.
- **Inheritance:** Inheritance is the process of deriving a new class from the existing ones.
- **Polymorphism:** It is a mechanism of representing objects having multiple forms used for different purposes.
- **Encapsulation:** It binds the data and the object together as a single unit, enabling tight coupling between them.

Use Case Diagram

Downloaded from EnggTree.com

EnggTree.com

A use case diagram is used to represent the dynamic behavior of a system. It encapsulates the system's functionality by incorporating use cases, actors, and their relationships. It models the tasks, services, and functions required by a system/subsystem of an application. It depicts the high-level functionality of a system and also tells how the user handles a system.

Purpose of Use Case Diagrams

The main purpose of a use case diagram is to portray the dynamic aspect of a system. It accumulates the system's requirement, which includes both internal as well as external influences. It invokes persons, use cases, and several things that invoke the actors and elements accountable for the implementation of use case diagrams. It represents how an entity from the external environment can interact with a part of the system.

Following are the purposes of a use case diagram given below:

1. It gathers the system's needs.
www.EnggTree.com
2. It depicts the external view of the system.
3. It recognizes the internal as well as external factors that influence the system.
4. It represents the interaction between the actors.

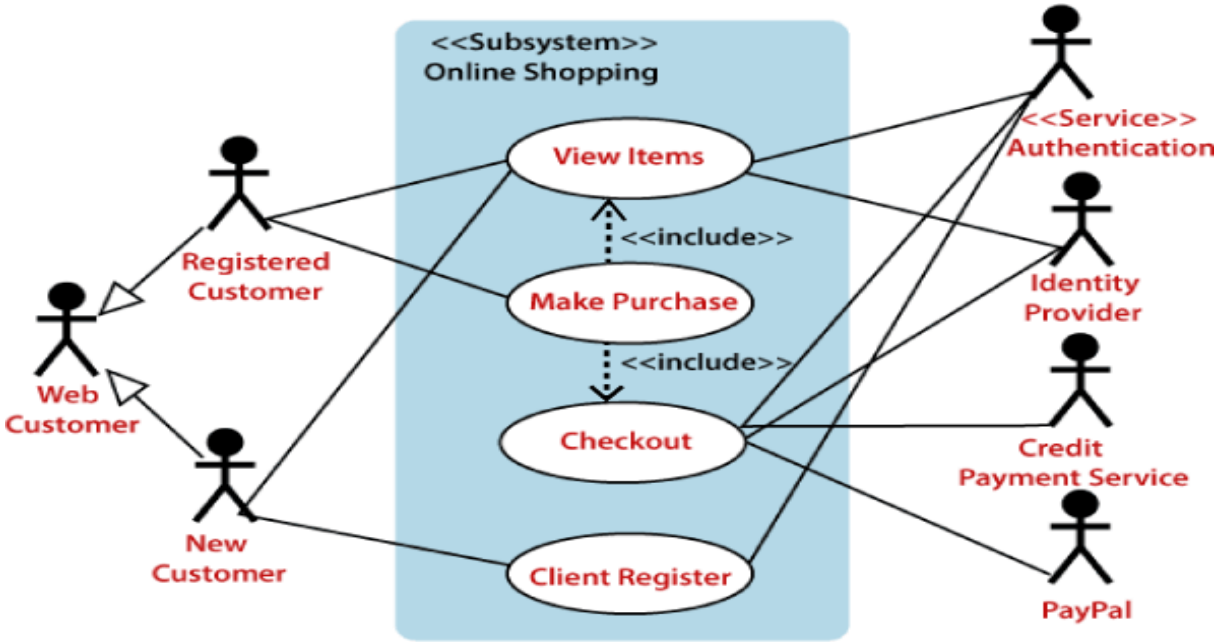
Example of a Use Case Diagram

A use case diagram depicting the Online Shopping website is given below.

Here the Web Customer actor makes use of any online shopping website to purchase online. The top-level uses are as follows; View Items, Make Purchase, Checkout, Client Register. The **View Items** use case is utilized by the customer who searches and view products. The **Client Register** use case allows the customer to register itself with the website for availing gift vouchers, coupons, or getting a private sale invitation. It is to be noted that the **Checkout** is an included use case, which is part of **Making Purchase,** and it is not available by itself.

Downloaded from EnggTree.com

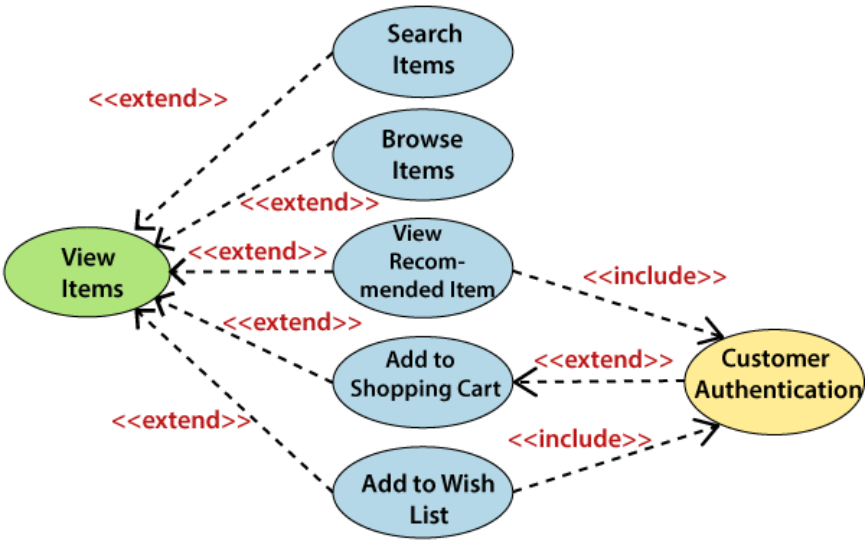
EnggTree.com



- The **View Items** is further extended by several use cases such as; Search Items, Browse Items, View Recommended Items, Add to Shopping Cart, Add to Wish list. All of these extended use cases provide some functions to customers, which allow them to search for an item.

www.EnggTree.com

- Both **View Recommended Item** and **Add to Wish List** include the Customer Authentication use case, as they necessitate authenticated customers, and simultaneously item can be added to the shopping cart without any user authentication.



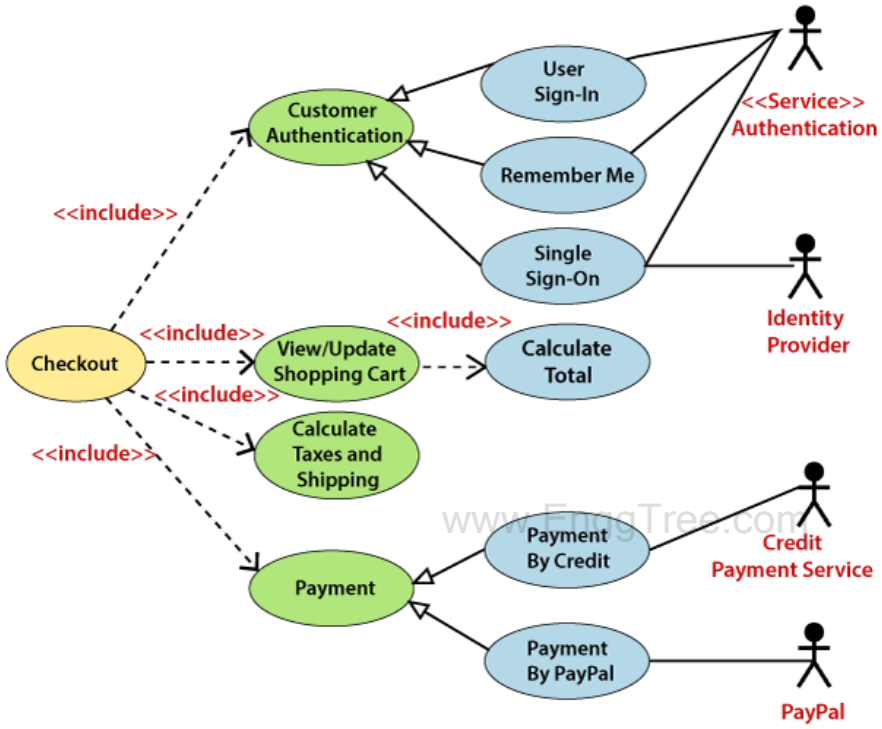
Similarly, the **Checkout** use case also includes the following use cases, as shown below. It requires an authenticated Web Customer, which can be done by login page, user authentication

Downloaded from EnggTree.com

EnggTree.com

- cookie ("Remember me"), or Single Sign-On (SSO). SSO needs an external identity provider's participation, while Web site authentication service is utilized in all these use cases.

- The Checkout use case involves Payment use case that can be done either by the credit cardand external credit payment services or with PayPal.



Downloaded from EnggTree.com

EnggTree.com

5. USE CASE MODEL

Following are some important tips that are to be kept in mind while drawing a use casediagram:

1. A simple and complete use case diagram should be articulated.
2. A use case diagram should represent the most significant interaction among the multipleinteractions.
3. At least one module of a system should be represented by the use case diagram.
4. If the use case diagram is large and more complex, then it should be drawn moregeneralized.

1. Class diagram

The class diagram depicts a static view of an application. It represents the types of objects residing in the system and the relationships between them. A class consists of its objects, and also it may inherit from other classes. A class diagram is used to visualize, describe, document various different aspects of the system, and also construct executable software code.

www.EnggTree.com

It shows the attributes, classes, functions, and relationships to give an overview of the softwaresystem. It constitutes class names, attributes, and functions in a separate compartment thathelps in software development. Since it is a collection of classes, interfaces, associations, collaborations, and constraints, it is termed as a structural diagram.

Purpose of Class Diagrams

The main purpose of class diagrams is to build a static view of an application. It is the only diagram that is widely used for construction, and it can be mapped with object-oriented languages. It is one of the most popular UML diagrams. Following are the purpose of class diagrams given below:

1. It analyses and designs a static view of an application.

Downloaded from EnggTree.com

EnggTree.com

2. It describes the major responsibilities of a system.
3. It is a base for component and deployment diagrams.
4. It incorporates forward and reverse engineering.

Benefits of Class Diagrams

1. It can represent the object model for complex systems.
2. It reduces the maintenance time by providing an overview of how an application isstructured before coding.
3. It provides a general schematic of an application for better understanding.
4. It represents a detailed chart by highlighting the desired code, which is to beprogrammed.
5. It is helpful for the stakeholders and the developers.

Vital components of a Class Diagram

The class diagram is made up of three sections:

www.EnggTree.com

o **Upper Section:** The upper section encompasses the name of the class. A class is a representation of similar objects that shares the same relationships, attributes, operations, and semantics. Some of the following rules that should be taken intoaccount while representing a class are given below:

1. Capitalize the initial letter of the class name.
2. Place the class name in the center of the upper section.
3. A class name must be written in bold format.
4. The name of the abstract class should be written in italics format.

b. **Middle Section:** The middle section constitutes the attributes, which describe the quality of the class. The attributes have the following characteristics:

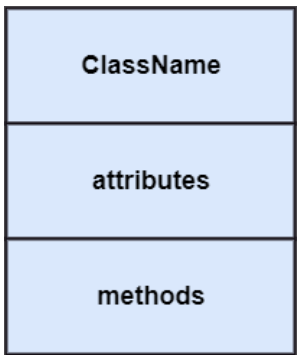
1. The attributes are written along with its visibility factors, which are public (+), private (-), protected (#), and package (~).
2. The accessibility of an attribute class is illustrated by the visibility factors.

Downloaded from EnggTree.com

EnggTree.com

3. A meaningful name should be assigned to the attribute, which will explain its usage inside the class.

b. **Lower Section:** The lower section contains methods or operations. The methods are represented in the form of a list, where each method is written in a single line. It demonstrates how a class interacts with data.



Relationships

In UML, relationships are of three types:

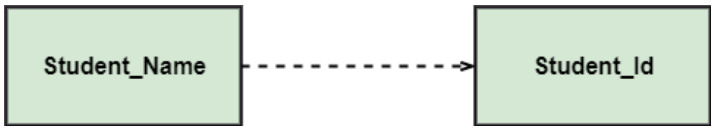
www.EnggTree.com

Downloaded from EnggTree.com

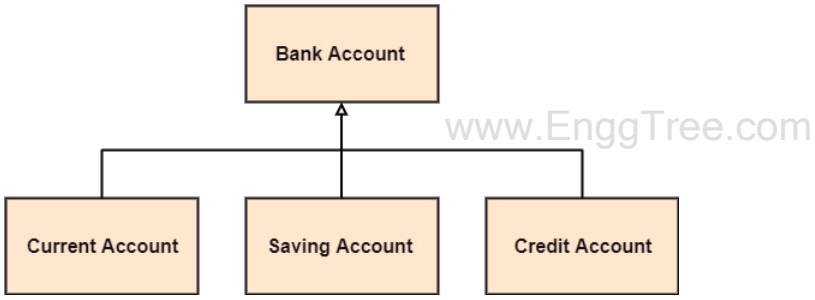
EnggTree.com

- **Dependency:** A dependency is a semantic relationship between two or more classes where a change in one class cause changes in another class. It forms a weaker relationship.

  In the following example, Student_Name is dependent on the Student_Id.



- **Generalization:** A generalization is a relationship between a parent class (superclass) and a child class (subclass). In this, the child class is inherited from the parent class. For example, The Current Account, Saving Account, and Credit Account are the generalized form of Bank Account.



www.EnggTree.com

- **Association:** It describes a static or physical connection between two or more objects. It depicts how many objects are there in the relationship. For example, a department is associated with the college.

Multiplicity: It defines a specific range of allowable instances of attributes. In case if a range is not specified, one is considered as a default multiplicity.

Class Diagram Example

Downloaded from EnggTree.com

EnggTree.com

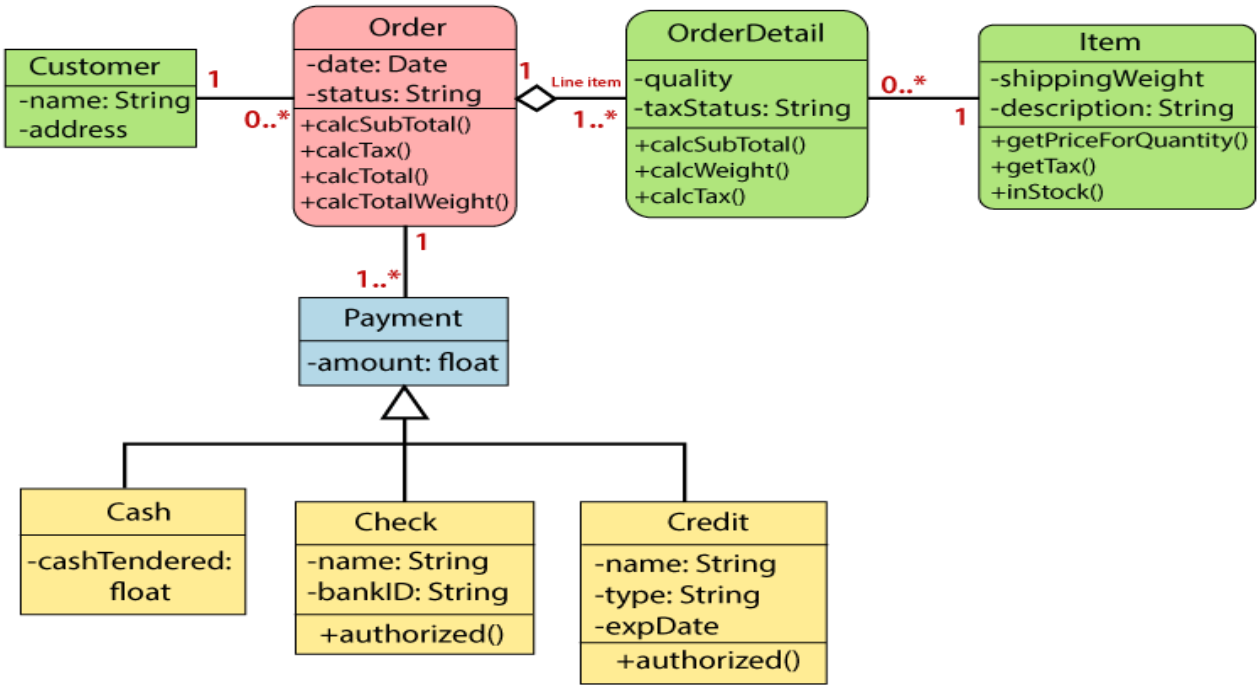
A class diagram describing the sales order system is given below.

24

www.EnggTree.com

Downloaded from EnggTree.com

EnggTree.com



Usage of Class diagrams

The class diagram is used to represent a static view of the system. It plays an essential role in the establishment of the component and deployment diagrams. It helps to construct an executable code to perform forward and backward engineering for any system, or we can say it is mainly used for construction. It represents the mapping with object-oriented languages that are C++, Java, etc. Class diagrams can be used for the following purposes:

1. To describe the static view of a system.
2. To show the collaboration among every instance in the static view.
3. To describe the functionalities performed by the system.
4. To construct the software application using object-oriented languages.

2. INTERACTION DIAGRAM

As the name suggests, the interaction diagram portrays the interactions between distinct entities present in the model. It amalgamates both the activity and sequence diagrams. The communication is nothing but units of the behaviour of a

Downloaded from EnggTree.com

EnggTree.com

classifier that provides context for interactions.

A set of messages that are interchanged between the entities to achieve certain specified tasks in the system is termed as interaction. It may incorporate any feature of the classifier of which it has access. In the interaction diagram, the critical component is the messages and the lifeline.

In UML, the interaction overview diagram initiates the interaction between the objects utilizing message passing. While drawing an interaction diagram, the entire focus is to represent the relationship among different objects which are available within the system boundary and the message exchanged by them to communicate with each other.

The message exchanged among objects is either to pass some information or to request some information. And based on the information, the interaction diagram is categorized into the sequence diagram, collaboration diagram, and timing diagram.

www.EnggTree.com

The sequence diagram envisions the order of the flow of messages inside the system by depicting the communication between two lifelines, just like a time-ordered sequence of events.

The collaboration diagram, which is also known as the communication diagram, represents how lifelines connect within the system, whereas the timing diagram focuses on that instant when a message is passed from one element to the other.

Downloaded from EnggTree.com

EnggTree.com

Purpose of an Interaction Diagram

The interaction diagram helps to envision the interactive (dynamic) behavior of any system. It portrays how objects residing in the system communicates and connects to each other. It also provides us with a context of communication between the lifelines inside the system.

Following are the purpose of an interaction diagram given below:

5. To visualize the dynamic behavior of the system.
6. To envision the interaction and the message flow in the system.
7. To portray the structural aspects of the entities within the system.
8. To represent the order of the sequenced interaction in the system.
9. To visualize the real-time data and represent the architecture of an object-orientedsystem.

www.EnggTree.com

How to draw an Interaction Diagram?

Since the main purpose of an interaction diagram is to visualize the dynamic behavior of the system, it is important to understand what a dynamic aspect really is and how we can visualize it. The dynamic aspect is nothing but a screenshot of the system at the run time.

Before drawing an interaction diagram, the first step is to discover the scenario for which the diagram will be made. Next, we will identify various lifelines that will be invoked in the communication, and then we will classify each lifeline. After that, the connections are investigated and how the lifelines are interrelated to each other.

Following are some things that are needed:

1. A total no of lifeline which will take part in the communication.

Downloaded from EnggTree.com

EnggTree.com

2. The sequence of the message flow among several entities within the system.
3. No operators used to ease out the functionality of the diagram.
4. Several distinct messages that depict the interactions in a precise and clear way.
5. The organization and structure of a system.
6. The order of the sequence of the flow of messages.
7. Total no of time constructs of an object.

Use of an Interaction Diagram

The interaction diagram can be used for:

1. The sequence diagram is employed to investigate a new application.
2. The interaction diagram explores and compares the use of the collaboration diagramsequence diagram and the timing diagram.
3. The interaction diagram represents the interactive (dynamic) behaviour of the system.
4. The sequence diagram portrays the order of control flow from one element to the other elements inside the system, whereas the collaboration diagrams are employed to get an overview of the object architecture of the system.
5. The interaction diagram models the system as a time-ordered sequence of a system.
6. The interaction diagram models the system as a time-ordered sequence of a system.
7. The interaction diagram systemizes the structure of the interactive elements.

3. ACTIVITY DIAGRAM

In UML, the activity diagram is used to demonstrate the flow of control within the system rather than the implementation. It models the concurrent and sequential activities.

The activity diagram helps in envisioning the workflow from one activity to another. It put emphasis on the condition of flow and the order in which it occurs.

Downloaded from EnggTree.com

EnggTree.com

The flow can be sequential, branched, or concurrent, and to deal with such kinds of flows, the activity diagram has come up with a fork, join, etc.

It is also termed as an object-oriented flowchart. It encompasses activities composed of a set of actions or operations that are applied to model the behavioural diagram.

Components of an Activity Diagram

Following are the component of an activity diagram:

Activities

The categorization of behaviour into one or more actions is termed as an activity. In other words, it can be said that an activity is a network of nodes that are connected by edges. The edges depict the flow of execution. It may contain action nodes, control nodes, or object nodes.

www.EnggTree.com

The control flow of activity is represented by control nodes and object nodes that illustrates the objects used within an activity. The activities are initiated at the initial node and are terminated at the final node.

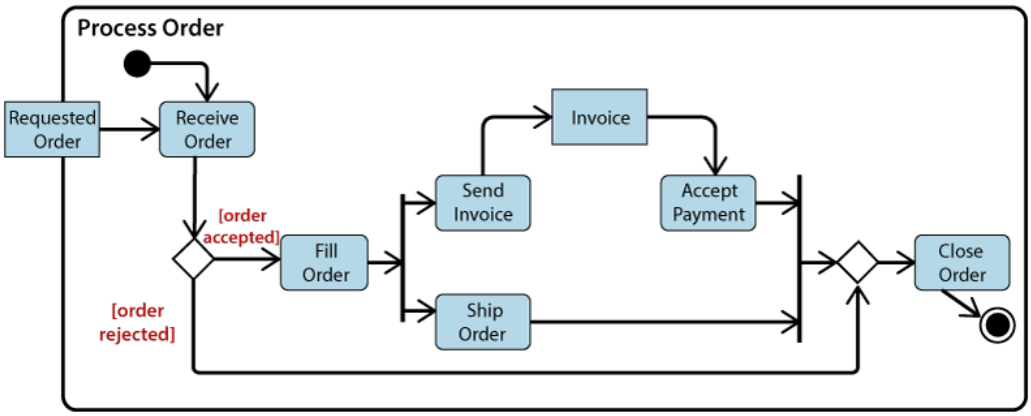
Example of an Activity Diagram

An example of an activity diagram showing the business flow activity of order processing is given below.

Here the input parameter is the Requested order, and once the order is accepted, all of the required information is then filled, payment is also accepted, and then the order is shipped. It permits order shipment before an invoice is sent or payment is completed.

Downloaded from EnggTree.com

EnggTree.com



When to use an Activity Diagram?

An activity diagram can be used to portray business processes and workflows. Also, it used for modeling business as well as the software. An activity diagram is utilized for the followings:

1. To graphically model the workflow in an easier and understandable way.
2. To model the execution flow among several activities.
3. To model comprehensive information of a function or an algorithm employed withinthe system.

Downloaded from EnggTree.com

EnggTree.com

CCS356-OBJECT ORIENTED SOFTWARE ENGINEERING

www.EnggTree.com

Downloaded from EnggTree.com

EnggTree.com

STATE CHART DIAGRAM

A **state diagram** is used to represent the condition of the system or part of the system at finite instances of time. It's a **behavioral** diagram and it represents the behavior using finite state transitions. State diagrams are also referred to as **State machines** and **State-chart Diagrams**. These terms are often used interchangeably. So simply, a state 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 diagrams. We prefer to model the states with three or more states.

Uses of state chart diagram –

- We use it to state the events responsible for change in state (we do not show whatprocesses cause those events).
- We use it to model the dynamic behavior of the system .
- To understand the reaction of objects/classes to internal or external stimuli. Firstly let us understand what are **Behavior diagrams**?

www.EnggTree.com

There are two types of diagrams in UML :

1. **Structure Diagrams –** Used to model the static structure of a system, for example- class diagram, package diagram, object diagram, deployment diagram etc.
2. **Behavior diagram –** Used to model the dynamic change in the system over time. They are used to model and construct the functionality of a system. So, a behavior diagram simply guides us through the functionality of the system using Use case diagrams, Interaction diagrams, Activity diagrams and State diagrams.

Basic components of a statechart diagram –

1. **Initial state –** We use a black filled circle represent the initial state of a System or aclass.

Downloaded from EnggTree.com

EnggTree.com

Figure – initial state notation

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 instate.

Figure – transition

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.

www.EnggTree.com

Figure – state notation

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 concurrentstates.

Downloaded from EnggTree.com

EnggTree.com

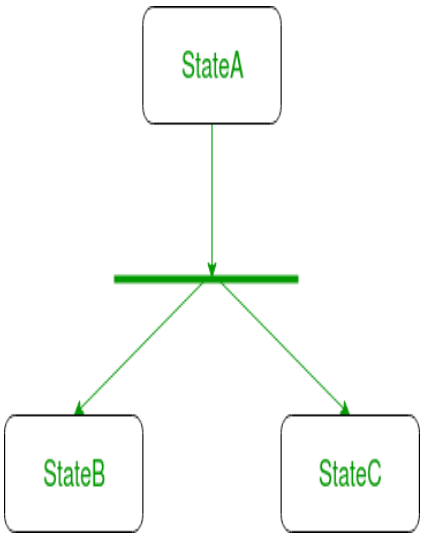


Figure – a diagram using the fork notation

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

www.EnggTree.com

state. We use the join notation when two or more states concurrently converge into one onthe occurrence of an event or events.

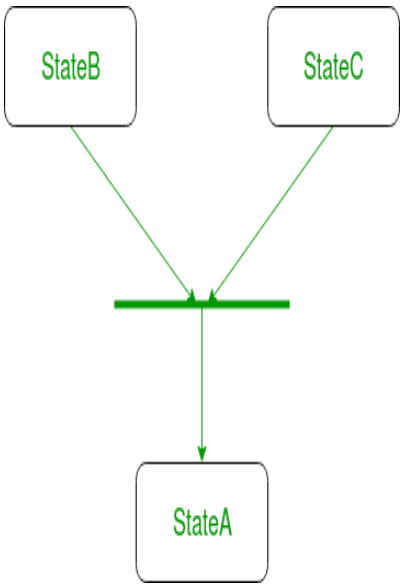


Figure – a diagram using join notation

6. **Self transition –** We use a solid arrow pointing back to the state itself to represent aself transition. There might be scenarios when the state of the

Downloaded from EnggTree.com

EnggTree.com

object does not change upon the occurrence of an event. We use self transitions to represent such cases.

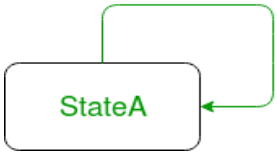


Figure – self transition notation

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.

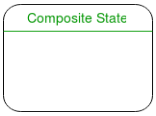


Figure – a state with internal activities

8. **Final state –** We use a filled circle within a circle notation to represent the final statein a state machine diagram.

www.EnggTree.com

Figure – final state notation

Steps to draw a state diagram –

1. Identify the initial state and the final terminating states.
2. Identify the possible states in which the object can exist (boundary values corresponding to different attributes guide us in identifying different states).
3. Label the events which trigger these transitions.

Example – state diagram for an online order –

Downloaded from EnggTree.com

EnggTree.com

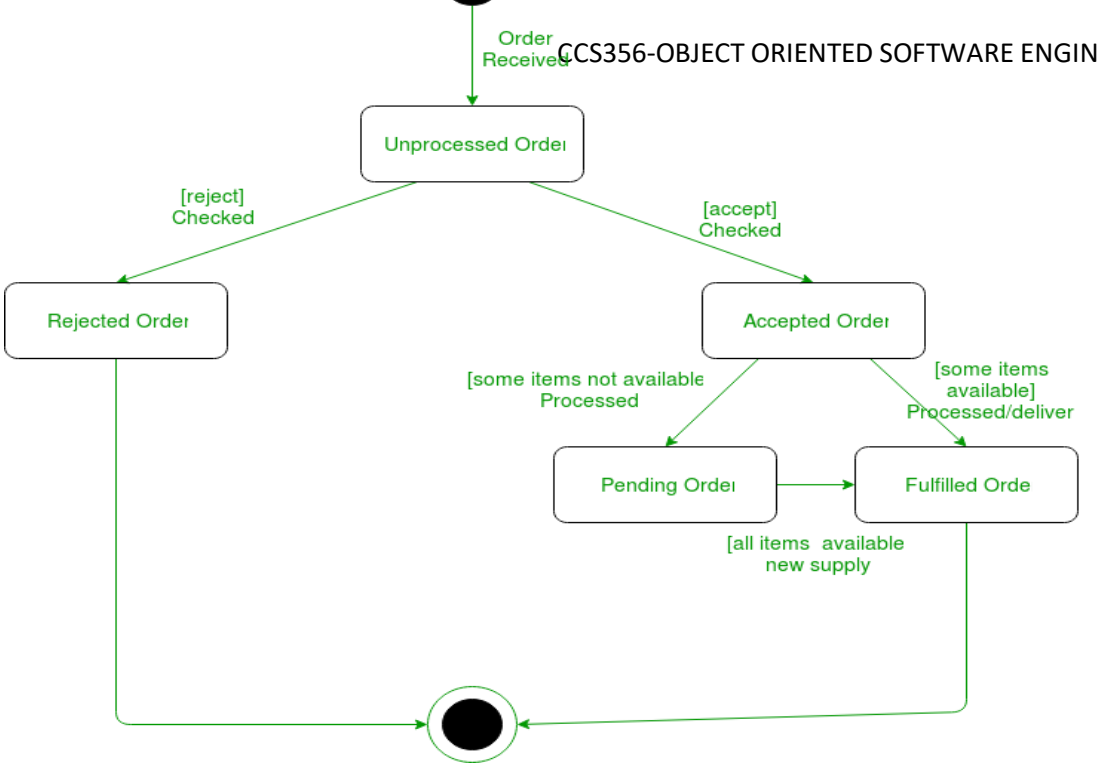


Figure – state diagram for an online order

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

1. On the event of an order being received, we transit from our initial state toUnprocessed 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 fulfilledorder state.
5. However if the items are not available we transit to the Pending Order state.

Functional Modelling

Functional Modelling gives the process perspective of the object-oriented analysis model and an overview of what the system is supposed to do. It defines the function of the internal processes in the system with the aid of Data Flow Diagrams (DFDs). It depicts the functional derivation of the data values without indicating how they are derived when they are computed, or why they need to be computed.

Downloaded from EnggTree.com

EnggTree.com

DATA FLOW DIAGRAMS

Functional Modelling is represented through a hierarchy of DFDs. The DFD is a graphical representation of a system that shows the inputs to the system, the processing upon the inputs, the outputs of the system as well as the internal data stores. DFDs illustrate the series of transformations or computations performed on the objects or the system, and the external controls and objects that affect the transformation.

Rumbaugh et al. have defined DFD as, "A data flow diagram is a graph which shows the flow of data values from their sources in objects through processes that transform them to their destinations on other objects."

The four main parts of a DFD are −

- Processes,
- Data Flows,
- Actors, and
- Data Stores.

The other parts of a DFD are −

- Constraints, and
- Control Flows.

www.EnggTree.com

Features of a

DFDProcesses

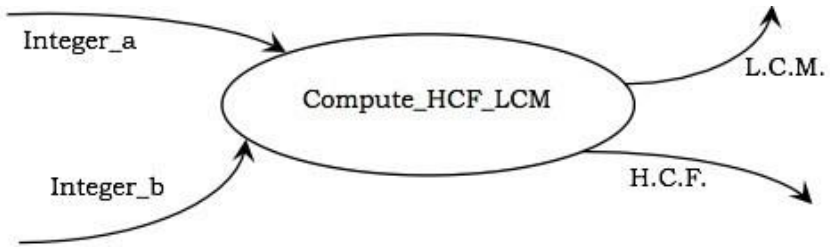
Processes are the computational activities that transform data values. A whole system can be visualized as a high-level process. A process may be further divided into smaller components.The lowest-level process may be a simple function.

Representation in DFD − A process is represented as an ellipse with its name written inside itand contains a fixed number of input and output data values.

Example − The following figure shows a process Compute_HCF_LCM that accepts twointegers as inputs and outputs their HCF (highest common factor) and LCM (least common multiple).

Downloaded from EnggTree.com

EnggTree.com



Data Flows

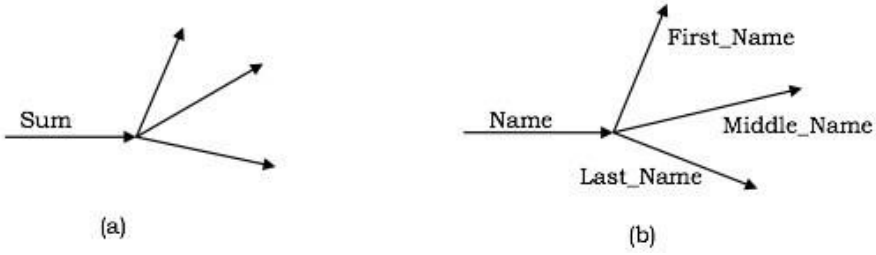
Data flow represents the flow of data between two processes. It could be between an actor and a process, or between a data store and a process. A data flow denotes the value of a data item atsome point of the computation. This value is not changed by the data flow.

Representation in DFD − A data flow is represented by a directed arc or an arrow, labelled with the name of the data item that it carries.

In the above figure, Integer_a and Integer_b represent the input data flows to the process, while
L.C.M. and H.C.F. are the output data flows.

A data flow may be forked in the following cases −

- The output value is sent to several places as shown in the following figure. Here, the output arrows are unlabelled as they denote the same value.
- The data flow contains an aggregate value, and each of the components is sent to different places as shown in the following figure. Here, each of the forked componentsis labelled.



Actors

Actors are the active objects that interact with the system by either producing data and inputting them to the system, or consuming data produced by the system. In other words, actorsserve as the sources and the sinks of data.

Representation in DFD − An actor is represented by a rectangle. Actors are connected to theinputs and outputs and lie on the boundary of the DFD.

Example − The following figure shows the actors, namely, Customer and Sales_Clerk in acounter sales system.

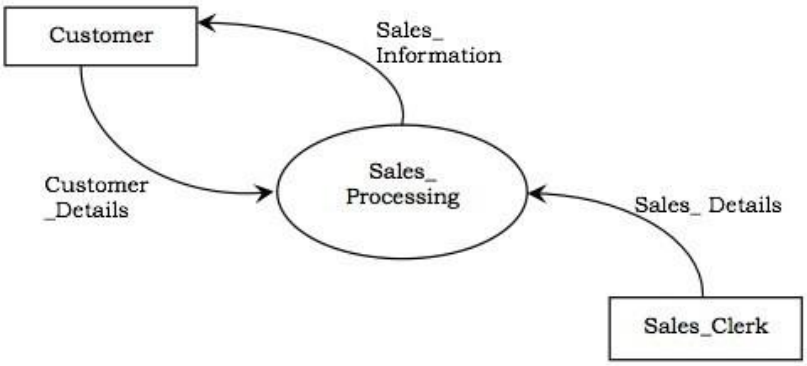
Downloaded from EnggTree.com

EnggTree.com

www.EnggTree.com

Downloaded from EnggTree.com

EnggTree.com



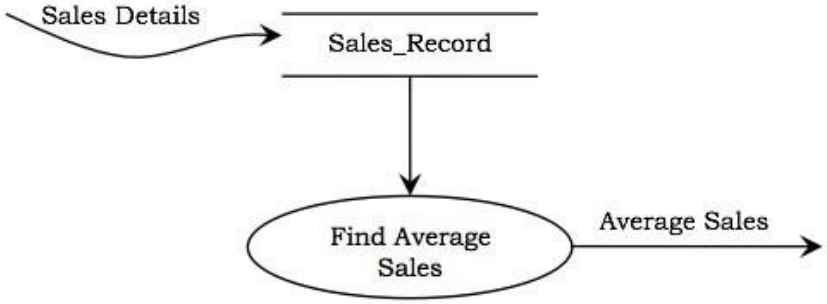
Data Stores

Data stores are the passive objects that act as a repository of data. Unlike actors, they cannot perform any operations. They are used to store data and retrieve the stored data. They representa data structure, a disk file, or a table in a database.

Representation in DFD − A data store is represented by two parallel lines containing the name of the data store. Each data store is connected to at least one process. Input arrows contain information to modify the contents of the data store, while output arrows contain information retrieved from the data store. When a part of the information is to be retrieved, the output arrowis labelled. An unlabelled arrow denotes full data retrieval. A two-way arrow implies both retrieval and update.

www.EnggTree.com

Example − The following figure shows a data store, Sales_Record, that stores the details of all sales. Input to the data store comprises of details of sales such as item, billing amount, date, etc. To find the average sales, the process retrieves the sales records and computes the average.



Constraints

Constraints specify the conditions or restrictions that need to be satisfied over time. They allowadding new rules or modifying existing ones. Constraints can appear in all the three

Downloaded from EnggTree.com

EnggTree.com

models of object-oriented analysis.

- In Object Modelling, the constraints define the relationship between objects. They may also define the relationship between the different values that an object may take at different times.
- In Dynamic Modelling, the constraints define the relationship between the states and events of different objects.
- In Functional Modelling, the constraints define the restrictions on the transformations and computations.

Advantages and Disadvantages of DFD

Advantages	Disadvantages
DFDs depict the boundaries of a system and hence are helpful in portraying the relationship between the external objects and the processes within the system.	DFDs take a long time to create, which may not be feasible for practical purposes.
They help the users to have a knowledge about the system.	DFDs do not provide any information about the time-dependent behavior, i.e., they do not specify when the transformations are done.
The graphical representation serves as a blueprint for the programmers to develop a system.	They do not throw any light on the frequency of computations or the reasons for computations.
DFDs provide detailed information about the system processes.	The preparation of DFDs is a complex process that needs considerable expertise. Also, it is difficult for a non-technical person to understand.
They are used as a part of the system documentation.	The method of preparation is subjective and leaves ample scope to be imprecise.

www.EnggTree.com

Downloaded from EnggTree.com

EnggTree.com

FUNCTIONAL MODELS

The Object Model, the Dynamic Model, and the Functional Model are complementary to eachother for a complete Object-Oriented Analysis.

- Object modelling develops the static structure of the software system in terms of objects. Thus it shows the "doers" of a system.
- Dynamic Modelling develops the temporal behavior of the objects in response to external events. It shows the sequences of operations performed on the objects.
- Functional model gives an overview of what the system should do.

Functional Model and Object Model

The four main parts of a Functional Model in terms of object model are −

- **Process** − Processes imply the methods of the objects that need to be implemented.
- **Actors** − Actors are the objects in the object model.
- **Data Stores** − These are either objects in the object model or attributes of objects.
- **Data Flows** − Data flows to or from actors represent operations on or by objects. Dataflows to or from data stores represent queries or updates.

Functional Model and Dynamic Model

The dynamic model states when the operations are performed, while the functional model states how they are performed and which arguments are needed. As actors are active objects, the dynamic model has to specify when it acts. The data stores are passive objects and they only respond to updates and queries; therefore, the dynamic model need not specify when they act.

Object Model and Dynamic Model

The dynamic model shows the status of the objects and the operations performed on the occurrences of events and the subsequent changes in states. The state of the object as a result ofthe changes is shown in the object model.

Downloaded from EnggTree.com

EnggTree.com

www.EnggTree.com

Downloaded from EnggTree.com

EnggTree.com

UNIT III
SOFTWARE DESIGN

Software design – Design process – Design concepts – Coupling – Cohesion – Functional independence – Design patterns – Model-view-controller – Publish-subscribe – Adapter – Command – Strategy – Observer – Proxy – Facade – Architectural styles – Layered - Client Server - Tiered - Pipe and filter- User interface design-Case Study.

1. Software design:

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 or phases of design:

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

Elements of a System

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

www.EnggTree.com

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 or phases of design:

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

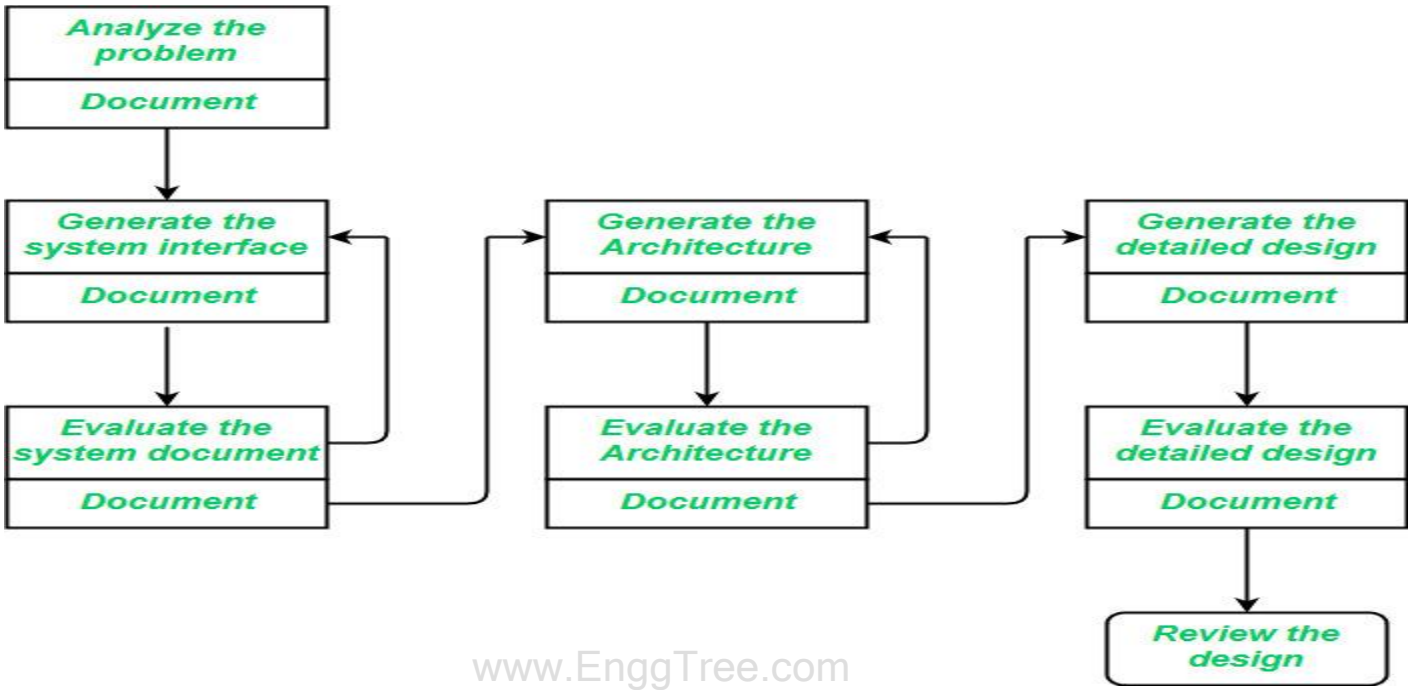
Elements of a System

1. **Architecture:** This is the conceptual model that defines the structure, behavior, and views of a system. We can use flowcharts to represent and illustrate the architecture.
2. **Modules:** These are components that handle one specific task in a system. A combination of the modules makes up the system.
3. **Components:** This provides a particular function or group of related functions. They are made up of modules.

Downloaded from EnggTree.com

EnggTree.com

4. **Interfaces:** This is the shared boundary across which the components of a system exchange information and relate.
5. **Data:** This is the management of the information and data flow.



www.EnggTree.com

Interface Design

Interface design is the specification of the interaction between a system and its environment. This phase proceeds at a high level of abstraction with respect to the inner workings of the system i.e, during interface design, the internal of the systems are completely ignored, and the system is treated as a black box. Attention is focused on the dialogue between the target system and the users, devices, and other systems with which it interacts. The design problem statement produced during the problem analysis step should identify the people, other systems, and devices which are collectively called agents.

Interface design should include the following details:

1. Precise description of events in the environment, or messages from agents to which the system must respond.
2. Precise description of the events or messages that the system must produce.
3. Specification of the data, and the formats of the data coming into and going out of the system.
4. 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:

Downloaded from EnggTree.com

EnggTree.com

1. Gross decomposition of the systems into major components.
2. Allocation of functional responsibilities to components.
3. Component Interfaces.
4. Component scaling and performance properties, resource consumption properties, reliability properties, and so forth.
5. 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:

1. Decomposition of major system components into program units.
2. Allocation of functional responsibilities to units.
3. User interfaces.
4. Unit states and state changes.
5. Data and control interaction between units.
6. Data packaging and implementation, including issues of scope and visibility of program elements.
7. Algorithms and data structures.

www.EnggTree.com

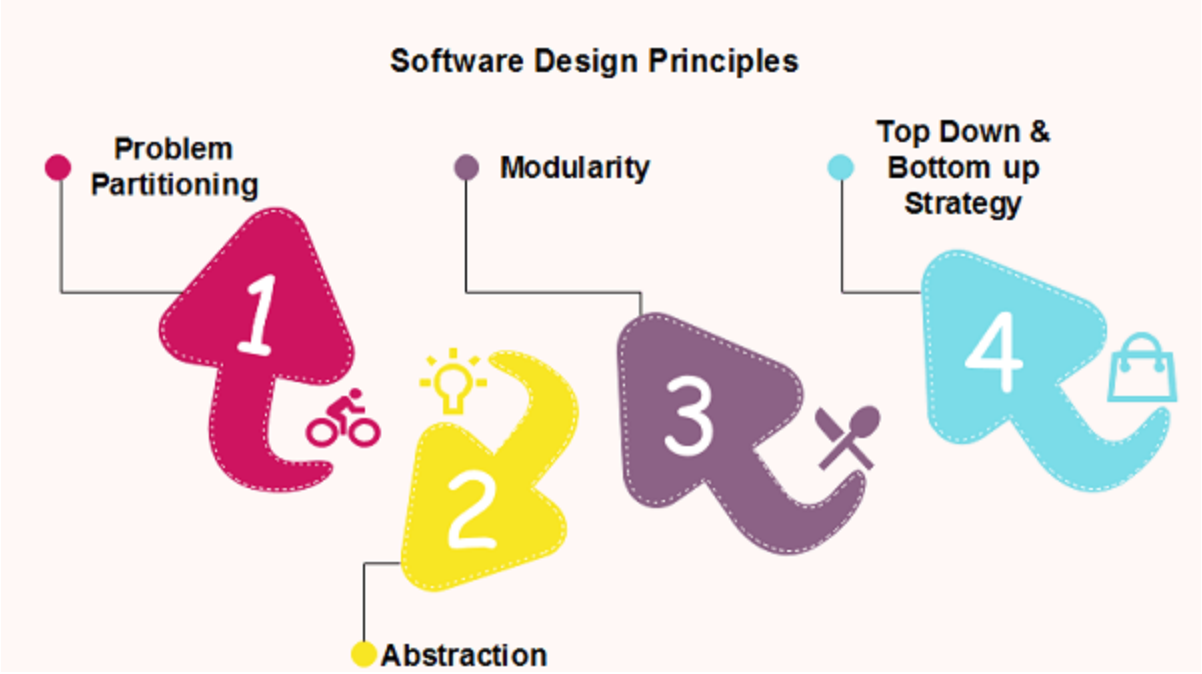
2. Software Design Principles

Software design principles are concerned with providing means to handle the complexity of the design process effectively. Effectively managing the complexity will not only reduce the effort needed for design but can also reduce the scope of introducing errors during design.

Following are the principles of Software Design

Downloaded from EnggTree.com

EnggTree.com



Problem Partitioning

For small problem, we can handle the entire problem at once but for the significant problem, divide the problems and conquer the problem it means to divide the problem into smaller pieces so that each piece can be captured separately.

For software design, the goal is to divide the problem into manageable pieces.

Benefits of Problem Partitioning

1. Software is easy to understand
2. Software becomes simple
3. Software is easy to test
4. Software is easy to modify
5. Software is easy to maintain
6. Software is easy to expand

Abstraction

An abstraction is a tool that enables a designer to consider a component at an abstract level without bothering about the internal details of the implementation. Abstraction can be used for existing element as well as the component being designed.

Downloaded from EnggTree.com

EnggTree.com

Here, there are two common abstraction mechanisms

1. Functional Abstraction
2. Data Abstraction

Functional Abstraction

i. A module is specified by the method it performs.

ii. The details of the algorithm to accomplish the functions are not visible to the user of the function.

Functional abstraction forms the basis for **Function oriented design approaches**.

Data Abstraction

Details of the data elements are not visible to the users of data. Data Abstraction forms the basis for **Object Oriented design approaches**.

Modularity

www.EnggTree.com

Modularity specifies to the division of software into separate modules which are differently named and addressed and are integrated later on in to obtain the completely functional software. It is the only property that allows a program to be intellectually manageable. Single large programs are difficult to understand and read due to a large number of reference variables, control paths, global variables, etc.

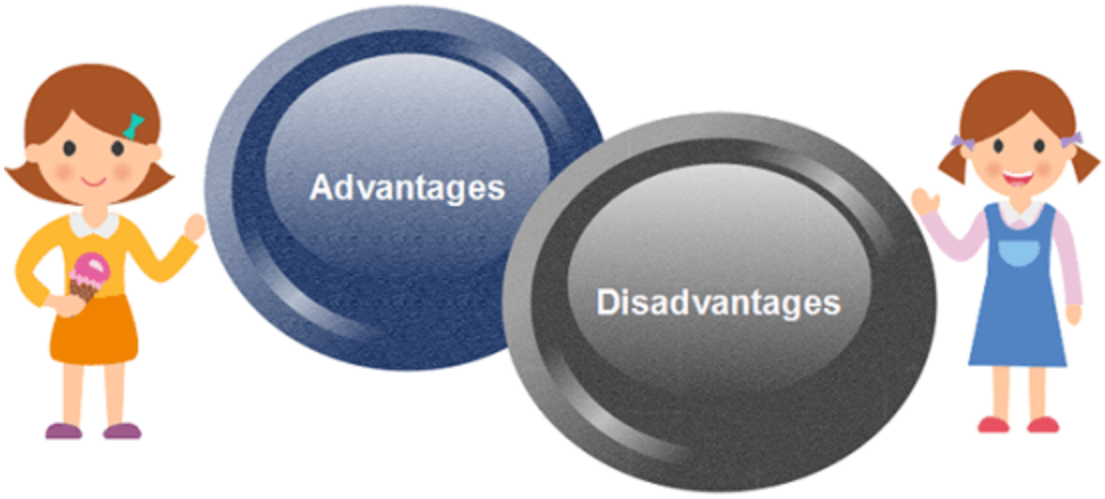
- ach module is a well-defined system that can be used with other applications.
- Each module has single specified objectives.
- Modules can be separately compiled and saved in the library.
- Modules should be easier to use than to build.
- Modules are simpler from outside than inside.

Advantages and Disadvantages of Modularity

In this topic, we will discuss various advantage and disadvantage of Modularity.

Downloaded from EnggTree.com

EnggTree.com



Advantages of Modularity

There are several advantages of Modularity

- It allows large programs to be written by several or different people
- It encourages the creation of commonly used routines to be placed in the library and used by other programs.
- It simplifies the overlay procedure of loading a large program into main storage.
- It provides more checkpoints to measure progress.
- It provides a framework for complete testing, more accessible to test
- It produced the well designed and more readable program.

www.EnggTree.com

Disadvantages of Modularity

There are several disadvantages of Modularity

- Execution time maybe, but not certainly, longer
- Storage size perhaps, but is not certainly, increased
- Compilation and loading time may be longer
- Inter-module communication problems may be increased
- More linkage required, run-time may be longer, more source lines must be written, and more documentation has to be done

Downloaded from EnggTree.com

EnggTree.com

3. 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.

www.EnggTree.com

- 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

Downloaded from EnggTree.com

EnggTree.com

- 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.

www.EnggTree.com

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

- The model of software is defined as a set of design classes.
- Every class describes the elements of problem domain and that focus on features of the problem which are user visible.

- OO design concept in Software Engineering
- Software design model elements

Downloaded from EnggTree.com

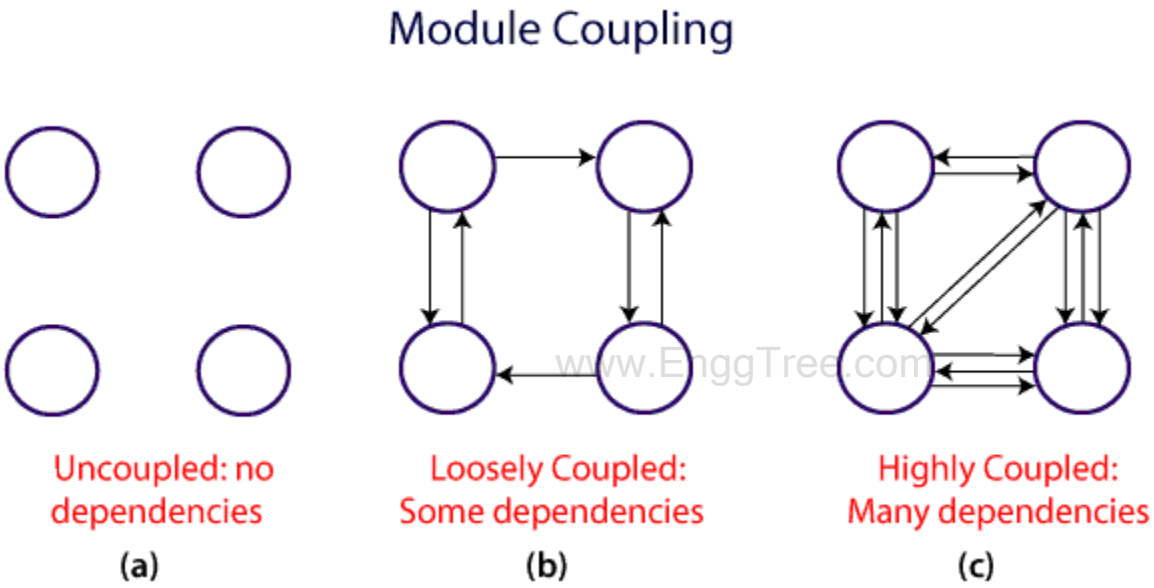
EnggTree.com

Coupling and Cohesion

Module Coupling

In software engineering, the coupling is the degree of interdependence between software modules. Two modules that are tightly coupled are strongly dependent on each other. However, two modules that are loosely coupled are not dependent on each other. **Uncoupled modules** have no interdependence at all within them.

The various types of coupling techniques are shown in fig:

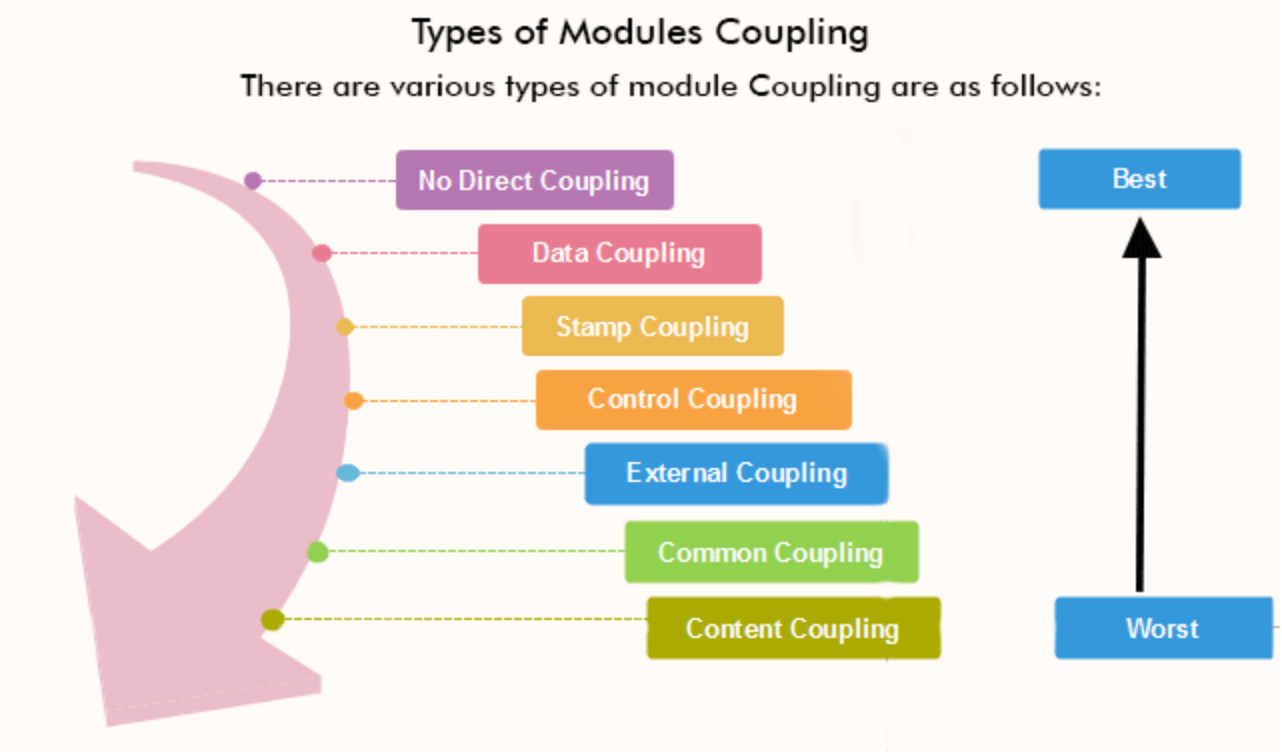


A good design is the one that has low coupling. Coupling is measured by the number of relations between the modules. That is, the coupling increases as the number of calls between modules increase or the amount of shared data is large. Thus, it can be said that a design with high coupling will have more errors.

Types of Module Coupling

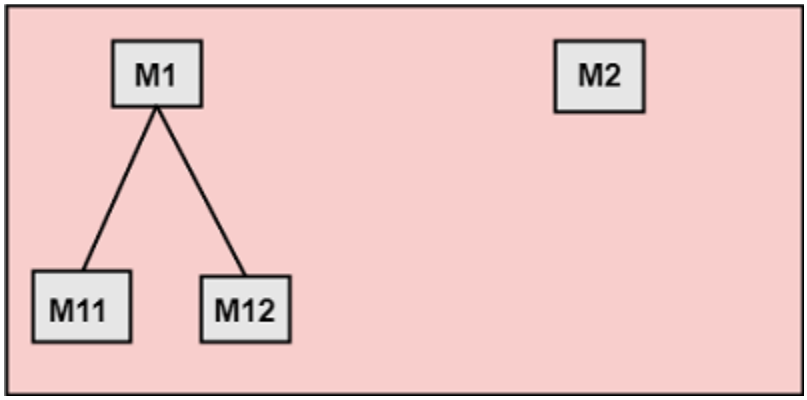
Downloaded from EnggTree.com

EnggTree.com



1. **No Direct Coupling:** There is no direct coupling between M1 and M2

www.EnggTree.com

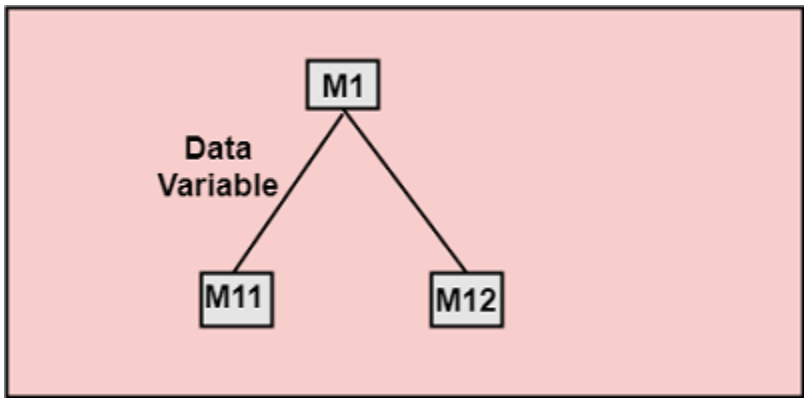


In this case, modules are subordinates to different modules. Therefore, no direct coupling.

2. Data Coupling: When data of one module is passed to another module, this is called data coupling.

Downloaded from EnggTree.com

EnggTree.com



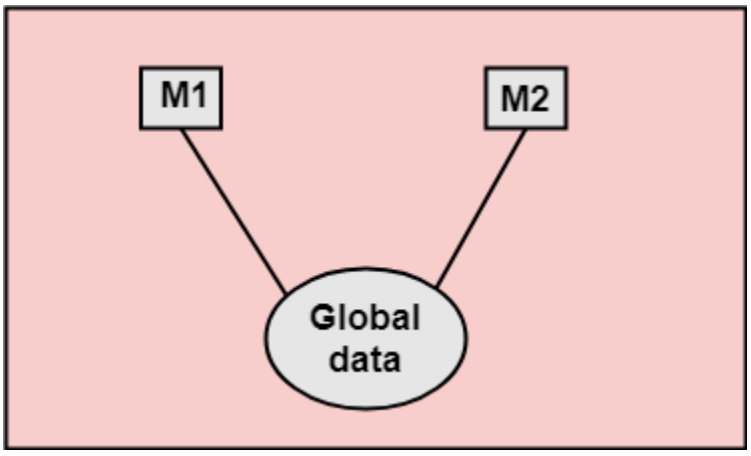
3. Stamp Coupling: Two modules are stamp coupled if they communicate using composite data items such as structure, objects, etc. When the module passes non-global data structure or entire structure to another module, they are said to be stamp coupled. For example, passing structure variable in C or object in C++ language to a module.

4. Control Coupling: Control Coupling exists among two modules if data from one module is used to direct the structure of instruction execution in another.

5. External Coupling: External Coupling arises when two modules share an externally imposed data format, communication protocols, or device interface. This is related to communication to external tools and devices.

www.EnggTree.com

6. Common Coupling: Two modules are common coupled if they share information through some global data items.



7. Content Coupling: Content Coupling exists among two modules if they share code, e.g., a branch from one module into another module.

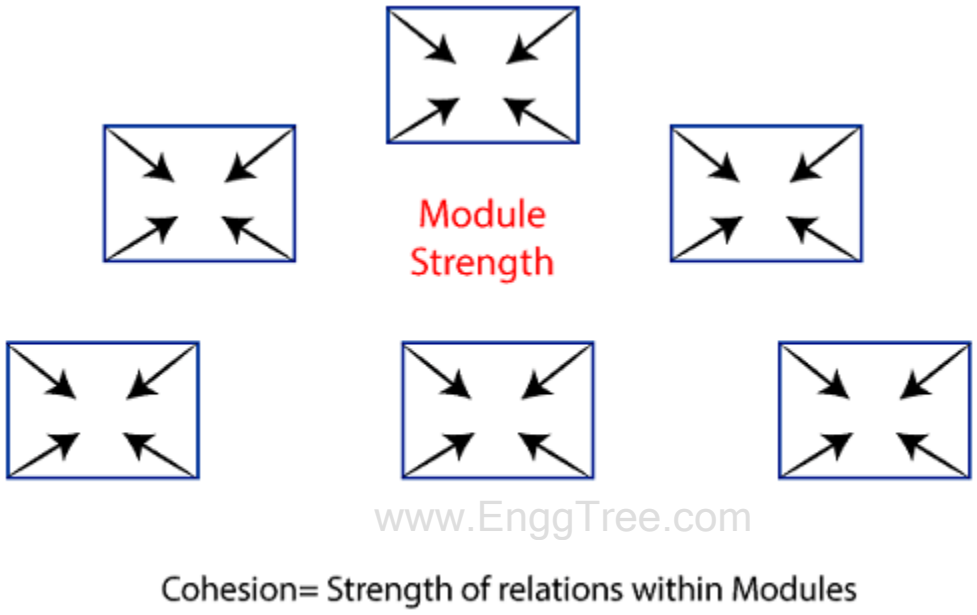
Downloaded from EnggTree.com

EnggTree.com

Module Cohesion

In computer programming, cohesion defines to the degree to which the elements of a module belong together. Thus, cohesion measures the strength of relationships between pieces of functionality within a given module. For example, in highly cohesive systems, functionality is strongly related.

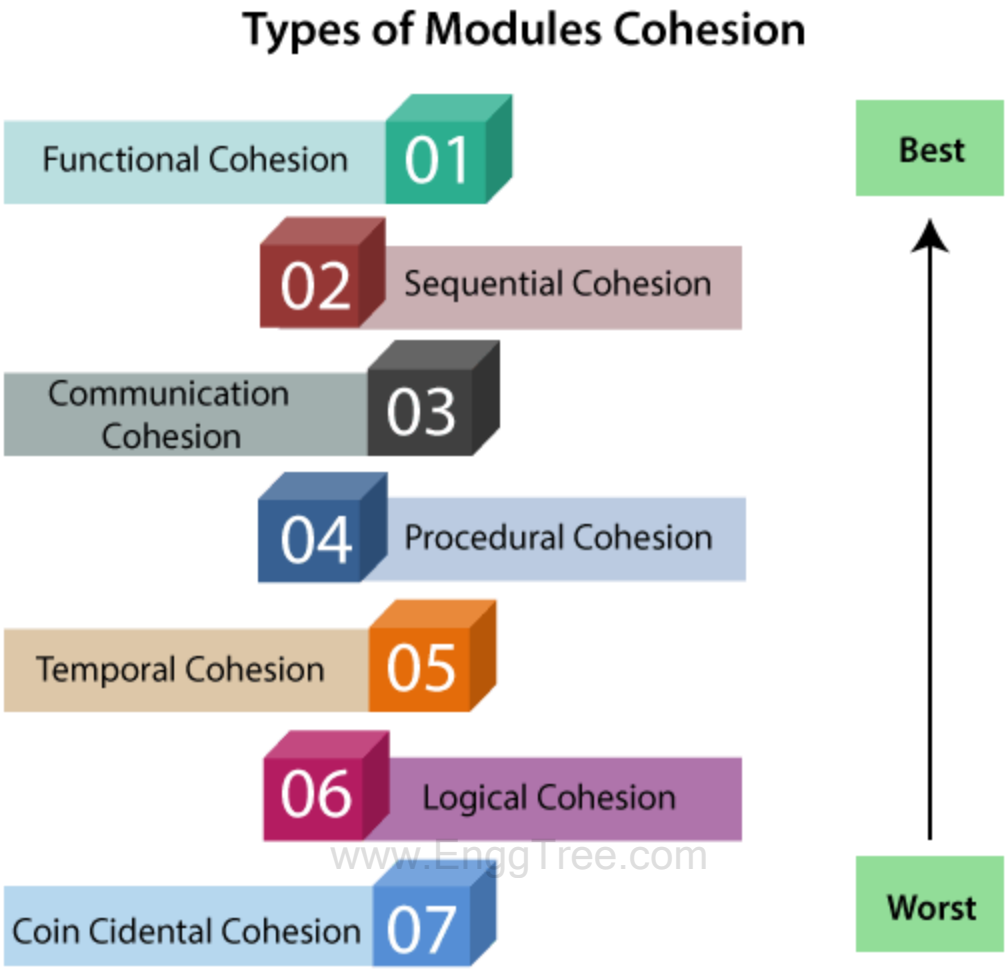
Cohesion is an **ordinal** type of measurement and is generally described as "high cohesion" or "low cohesion."



Types of Modules Cohesion

Downloaded from EnggTree.com

EnggTree.com



1. **Functional Cohesion:** Functional Cohesion is said to exist if the different elements of a module, cooperate to achieve a single function.
2. **Sequential Cohesion:** A module is said to possess sequential cohesion if the element of a module form the components of the sequence, where the output from one component of the sequence is input to the next.
3. **Communicational Cohesion:** A module is said to have communicational cohesion, if all tasks of the module refer to or update the same data structure, e.g., the set of functions defined on an array or a stack.
4. **Procedural Cohesion:** A module is said to be procedural cohesion if the set of purpose of the module are all parts of a procedure in which particular sequence of steps has to be carried out for achieving a goal, e.g., the algorithm for decoding a message.
5. **Temporal Cohesion:** When a module includes functions that are associated by the fact that all the methods must be executed in the same time, the module is said to exhibit temporal cohesion.

Downloaded from EnggTree.com

EnggTree.com

6. **Logical Cohesion:** A module is said to be logically cohesive if all the elements of the module perform a similar operation. For example Error handling, data input and data output, etc.
7. **Coincidental Cohesion:** A module is said to have coincidental cohesion if it performs a set of tasks that are associated with each other very loosely, if at all.

Differentiate between Coupling and Cohesion

Coupling	Cohesion
Coupling is also called Inter-Module Binding.	Cohesion is also called Intra-Module Binding.
Coupling shows the relationships between modules.	Cohesion shows the relationship within the module.
Coupling shows the relative **independence** between the modules.	Cohesion shows the module's relative **functional** strength.
While creating, you should aim for low coupling, i.e., dependency among modules should be less.	While creating you should aim for high cohesion, i.e., a cohesive component/ module focuses on a single function (i.e., single-mindedness) with little interaction with other modules of the system.
In coupling, modules are linked to the other modules.	In cohesion, the module focuses on a single thing.

www.EnggTree.com

Downloaded from EnggTree.com

EnggTree.com

FUNCTIONAL INDEPENDENCE IN SOFTWARE ENGINEERING

Functional independence in software engineering means that when a module focuses on a single task, it should be able to accomplish it with very little interaction with other modules.

In software engineering, if a module is functionally independent of other module then it means it has high cohesion and low coupling.

Functional independence is essential for good software design.

Example of functional independence

We will take an example of simple college level project to *explain concept of functional independence*.

Suppose you and your friends are asked to work on a calculator project as a team. Here we need to develop each calculator functionality in form of modules taking two user inputs.

www.EnggTree.com

So our modules are addition Module, subtraction Module, division Module, multiplication Module. Each one of you pick up one module for development purpose.

Before you enter into development phase, you and your team needs to make sure to design the project in such a way that each of the module that you develop individually should be able to perform its assigned task without requiring much or no interaction with your friends module.

What I intend to say is, if you are working on addition Module then your module should be able to independently perform addition operation on receiving user input. It should not require to make any interaction with other modules like subtraction Module, multiplication Module or etc.

This is actually the concept of having module as *functional independence* of other modules. There is an advantage of functional independence in software engineering which we are going to discuss next.

Advantage of functional independence

Downloaded from EnggTree.com

EnggTree.com

Advantages of functional independence are given below:

Error isolation

When a module is functionally independent then it performs most of its task independently without interacting with other modules much. This reduces the chances of error getting propagated to other modules. This helps in easily isolating and tracing the error.

Module reusability

A functionally independent module performs some well defined and specific task. So it becomes easy to reuse such modules in different program requiring same functionality.

Understandability

A functionally independent module is less complex so easy to understand. Since such modules are less interaction with other modules so can be understood in isolation.

www.EnggTree.com

Downloaded from EnggTree.com

EnggTree.com

Design Patterns

In software engineering, a **design pattern** is a general repeatable solution to a commonly occurring problem in software design. A design pattern isn't a finished design that can be transformed directly into code. It is a description or template for how to solve a problem that can be used in many different situations.

Uses of Design Patterns

Design patterns can speed up the development process by providing tested, proven development paradigms. Effective software design requires considering issues that may not become visible until later in the implementation. Reusing design patterns helps to prevent subtle issues that can cause major problems and improves code readability for coders and architects familiar with the patterns.

www.EnggTree.com

Often, people only understand how to apply certain software design techniques to certain problems. These techniques are difficult to apply to a broader range of problems. Design patterns provide general solutions, documented in a format that doesn't require specifics tied to a particular problem.

In addition, patterns allow developers to communicate using well-known, well understood names for software interactions. Common design patterns can be improved over time, making them more robust than ad-hoc designs.

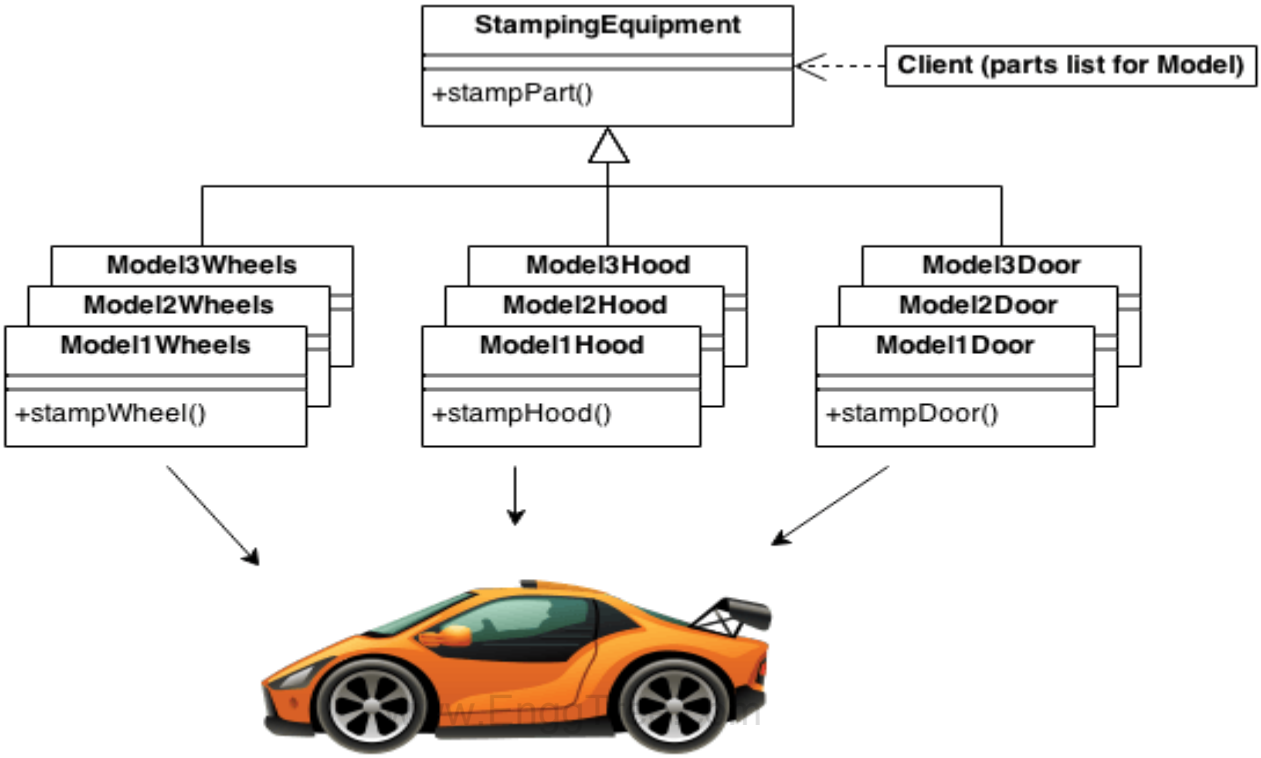
Creational design patterns

These design patterns are all about class instantiation. This pattern can be further divided into class-creation patterns and object-creational patterns. While

Downloaded from EnggTree.com

EnggTree.com

class-creation patterns use inheritance effectively in the instantiation process, object-creation patterns use delegation effectively to get the job done.



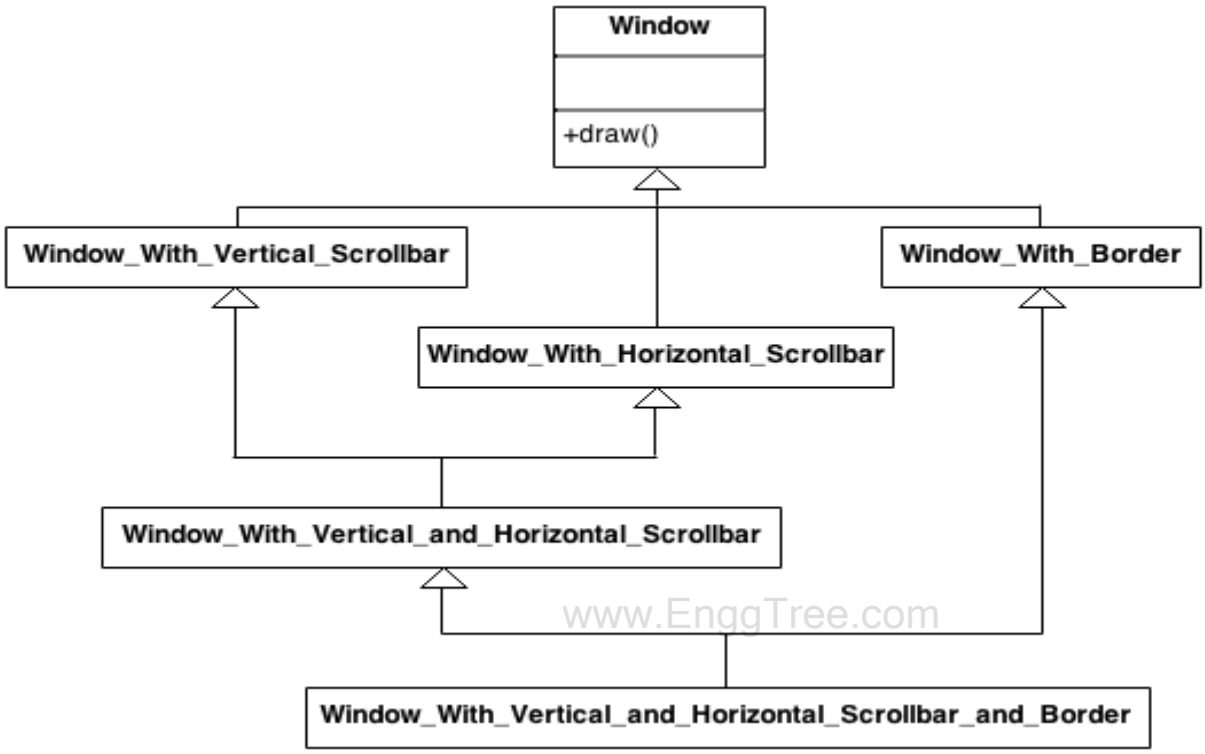
- **AbstractFactory**
  Creates an instance of several families of classes
- **Builder**
  Separates object construction from its representation
- **FactoryMethod**
  Creates an instance of several derived classes
- **ObjectPool**
  Avoid expensive acquisition and release of resources by recycling objects that are no longer in use
- **Prototype**
  A fully initialized instance to be copied or cloned
- **Singleton**
  A class of which only a single instance can exist

Downloaded from EnggTree.com

EnggTree.com

Structural design patterns

These design patterns are all about Class and Object composition. Structural class-creation patterns use inheritance to compose interfaces. Structural object-patterns define ways to compose objects to obtain new functionality

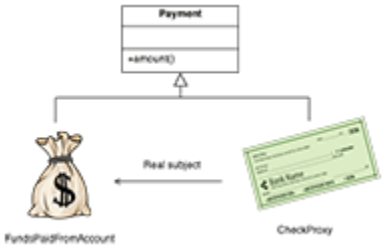


ty.

- **Adapter**
  Match interfaces of different classes
- **Bridge**
  Separates an object's interface from its implementation
- **Composite**
  A tree structure of simple and composite objects
- **Decorator**
  Add responsibilities to objects dynamically
- **Facade**
  A single class that represents an entire subsystem

Downloaded from EnggTree.com

EnggTree.com

- **Flyweight**
  A fine-grained instance used for efficient sharing
- 

  PrivateClassData
  Restricts accessor/mutator access

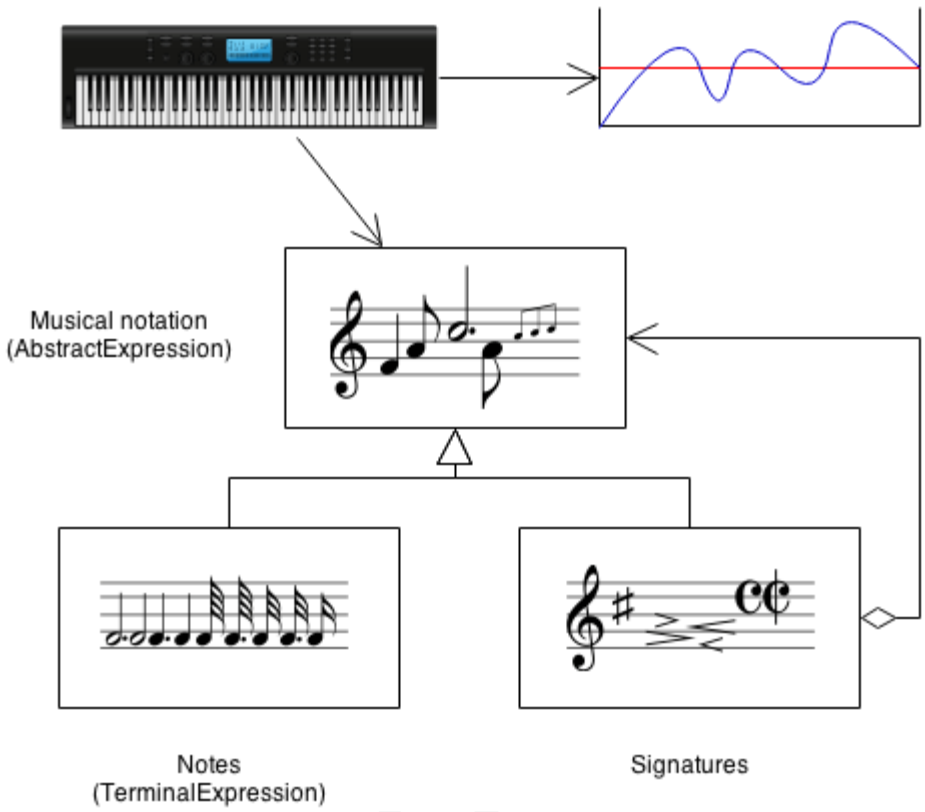
- **Proxy**
  An object representing another object

Behavioral design patterns

These design patterns are all about Class's objects communication. Behavioral patterns are those patterns that are most specifically concerned with communication between objects.

www.EnggTree.com

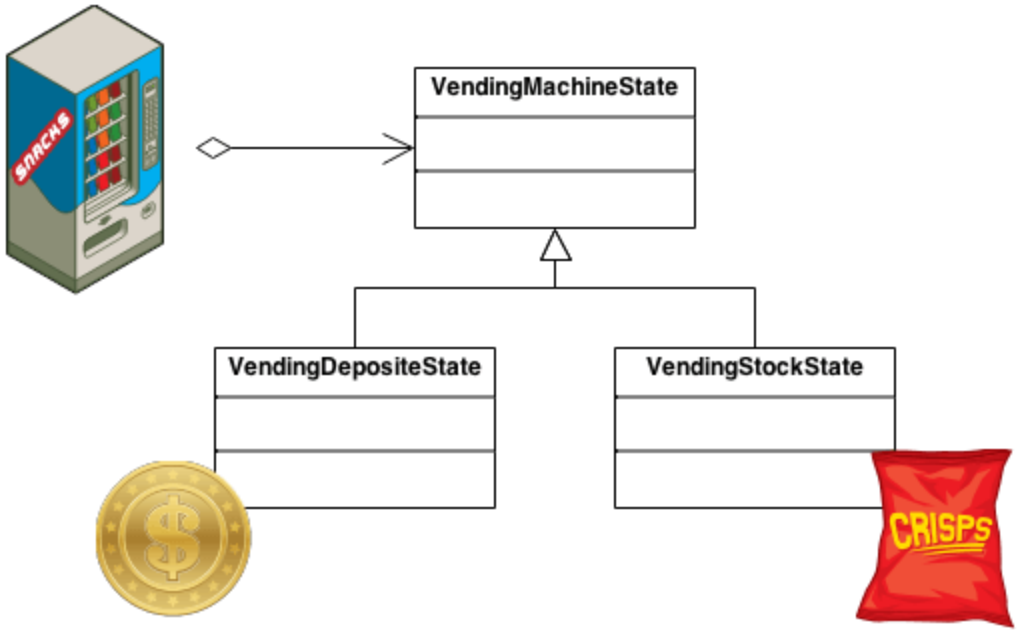
Downloaded from EnggTree.com



www.EnggTree.com

- **Chainofresponsibility**
  A way of passing a request between a chain of objects
- **Command**
  Encapsulate a command request as an object
- **Interpreter**
  A way to include language elements in a program
- **Iterator**
  Sequentially access the elements of a collection
- **Mediator**
  Defines simplified communication between classes
- **Memento**
  Capture and restore an object's internal state
- **NullObject**
  Designed to act as a default value of an object
- **Observer**
  A way of notifying change to a number of classes

Downloaded from EnggTree.com



- **State**
  Alter an object's behavior when its state changes

www.EnggTree.com

- **Strategy**
  Encapsulates an algorithm inside a class
- **Templatemethod**
  Defer the exact steps of an algorithm to a subclass
- **Visitor**
  Defines a new operation to a class without change

Types of Design Patterns

There are three types of Design Patterns,

- Creational Design Pattern
- Structural Design Pattern
- Behavioral Design Pattern

Creational Design Pattern

Creational Design Pattern abstract the instantiation process. They help in making a system independent of how its objects are created, composed and represented.

Importance of Creational Design Patterns:

- A class creational Pattern uses inheritance to vary the class that's instantiated, whereas an object creational pattern will delegate instantiation to another object.
- Creational patterns become important as systems evolve to depend more on object composition than class inheritance. As that happens, emphasis shifts away from hardcoding a

Downloaded from EnggTree.com

EnggTree.com

fixed set of behaviors toward defining a smaller set of fundamental behaviors that can be composed into any number of more complex ones.
- Creating objects with particular behaviors requires more than simply instantiating a class.

When to ue Creational Design Patterns

- **Complex Object Creation:** Use creational patterns when the process of creating an object is complex, involving multiple steps, or requires the configuration of various parameters.
- **Promoting Reusability:** Creational patterns promote object creation in a way that can be reused across different parts of the code or even in different projects, enhancing modularity and maintainability.
- **Reducing Coupling:** Creational patterns can help reduce the coupling between client code and the classes being instantiated, making the system more flexible and adaptable to changes.
- **Singleton Requirements:** Use the Singleton pattern when exactly one instance of a class is needed, providing a global point of access to that instance.
- **Step-by-Step Construction:** Builder pattern of creational design patterns is suitable when you need to construct a complex object step by step, allowing for the creation of different representations of the same object.

Advantages of Creational Design Patterns

- **Flexibility and Adaptability:** Creational patterns make it easier to introduce new types of objects or change the way objects are created without modifying existing client code. This enhances the system's flexibility and adaptability to change.
- **Reusability:** By providing a standardized way to create objects, creational patterns promote code reuse across different parts of the application or even in different projects. This leads to more maintainable and scalable software.
- **Centralized Control:** Creational patterns, such as Singleton and Factory patterns, allow for centralized control over the instantiation process. This can be advantageous in managing resources, enforcing constraints, or ensuring a single point of access.
- **Scalability:** With creational patterns, it's easier to scale and extend a system by adding new types of objects or introducing variations without causing major disruptions to the existing codebase.
- **Promotion of Good Design Practices:** Creational patterns often encourage adherence to good design principles such as abstraction, encapsulation, and the separation of concerns. This leads to cleaner, more maintainable code.

Disadvantages of Creational Design Patterns

- **Increased Complexity:** Introducing creational patterns can sometimes lead to increased complexity in the codebase, especially when dealing with a large number of classes, interfaces, and relationships.
- **Overhead:** Using certain creational patterns, such as the Abstract Factory or Prototype pattern, may introduce overhead due to the creation of a large number of classes and interfaces.
- **Dependency on Patterns:** Over-reliance on creational patterns can make the codebase dependent on a specific pattern, making it challenging to adapt to changes or switch to alternative solutions.

Downloaded from EnggTree.com

- **Readability and Understanding:** The use of certain creational patterns might make the code less readable and harder to understand, especially for developers who are not familiar with the specific pattern being employed.

Structural Design Patterns

Structural patterns are concerned with how classes and objects are composed to form larger structures. Structural class patterns use inheritance to compose interfaces or implementations.

Importance of Structural Design Patterns

- This pattern is particularly useful for making independently developed class libraries work together.
- Structural object patterns describe ways to compose objects to realize new functionality.
- It added flexibility of object composition comesfrom the ability to change the composition at run-time, which is impossible with static class composition.

When to ue Structural Design Patterns

- **Adapting to Interfaces:** Use structural patterns like the Adapter pattern when you need to make existing classes work with others without modifying their source code. This is particularly useful when integrating with third-party libraries or legacy code.
- **Organizing Object Relationships:** Structural patterns such as the Decorator pattern are useful when you need to add new functionalities to objects by composing them in a flexible and reusable way, avoiding the need for subclassing.
- **Simplifying Complex Systems:** When dealing with complex systems, structural patterns like the Facade pattern can be used to provide a simplified and unified interface to a set of interfaces in a subsystem.
- **Managing Object Lifecycle:** The Proxy pattern is helpful when you need to control access to an object, either for security purposes, to delay object creation, or to manage the object's lifecycle.
- **Hierarchical Class Structures:** The Composite pattern is suitable when dealing with hierarchical class structures where clients need to treat individual objects and compositions of objects uniformly.

www.EnggTree.com

Advantages of Structural Design Patterns

- **Flexibility and Adaptability:** Structural patterns enhance flexibility by allowing objects to be composed in various ways. This makes it easier to adapt to changing requirements without modifying existing code.
- **Code Reusability:** These patterns promote code reuse by providing a standardized way to compose objects. Components can be reused in different contexts, reducing redundancy and improving maintainability.
- **Improved Scalability:** As systems grow in complexity, structural patterns provide a scalable way to organize and manage the relationships between classes and objects. This supports the growth of the system without causing a significant increase in complexity.
- **Simplified Integration:** Structural patterns, such as the Adapter pattern, facilitate the integration of existing components or third-party libraries by providing a standardized interface. This makes it easier to incorporate new functionalities into an existing system.

Downloaded from EnggTree.com

EnggTree.com

- **Easier Maintenance:** By promoting modularity and encapsulation, structural patterns contribute to easier maintenance. Changes to one part of the system are less likely to affect other parts, reducing the risk of unintended consequences.
- **Solves Recurring Design Problems:** These patterns encapsulate solutions to recurring design problems. By applying proven solutions, developers can focus on higher-level design challenges unique to their specific applications.

Disadvantages of Structural Design Patterns

- **Complexity:** Introducing structural patterns can sometimes lead to increased complexity in the codebase, especially when multiple patterns are used or when dealing with a large number of classes and interfaces.
- **Overhead:** Some structural patterns, such as the Composite pattern, may introduce overhead due to the additional layers of abstraction and complexity introduced to manage hierarchies of objects.
- **Maintenance Challenges:** Changes to the structure of classes or relationships between objects may become more challenging when structural patterns are heavily relied upon. Modifying the structure may require updates to multiple components.
- **Limited Applicability:** Not all structural patterns are universally applicable. The suitability of a pattern depends on the specific requirements of the system, and using a pattern in the wrong context may lead to unnecessary complexity.

Behavioral Design Pattern

Behavioral patterns are concerned with algorithms and the assignment of responsibilities between objects. Behavioral patterns describe not just patterns of objects or classes but also the patterns of communication between them.

Importance of Behavioral Design Pattern

- These patterns characterize complex control flow that's difficult to follow at run-time.
- They shift focus away from flow of control to let you concentratejust on the way objects are interconnected.
- Behavioral class patterns use inheritance to distribute behavior between classes.

When to ue Behavioral Design Patterns

- **Communication Between Objects:** Use behavioral patterns when you want to define how objects communicate, collaborate, and interact with each other in a flexible and reusable way.
- **Encapsulation of Behavior:** Apply behavioral patterns to encapsulate algorithms, strategies, or behaviors, allowing them to vary independently from the objects that use them. This promotes code reusability and maintainability.
- **Dynamic Behavior Changes:** Use behavioral patterns when you need to allow for dynamic changes in an object's behavior at runtime without altering its code. This is particularly relevant for systems that require flexibility in behavior.
- **State-Dependent Behavior:** State pattern is suitable when an object's behavior depends on its internal state, and the object needs to change its behavior dynamically as its state changes.
- **Interactions Between Objects:** Behavioral patterns are valuable when you want to model and manage interactions between objects in a way that is clear, modular, and easy to understand.

Downloaded from EnggTree.com

EnggTree.com

Advantages of Behavioral Design Patterns

Flexibility and Adaptability:

- Behavioral patterns enhance flexibility by allowing objects to interact in a more dynamic and adaptable way. This makes it easier to modify or extend the behavior of a system without altering existing code.
- **Code Reusability:**
- Behavioral patterns promote code reusability by encapsulating algorithms, strategies, or behaviors in separate objects. This allows the same behavior to be reused across different parts of the system.
- **Separation of Concerns:**
- These patterns contribute to the separation of concerns by dividing the responsibilities of different classes, making the codebase more modular and easier to understand.
- **Encapsulation of Algorithms:**
- Behavioral patterns encapsulate algorithms, strategies, or behaviors in standalone objects, making it possible to modify or extend the behavior without affecting the client code.
- **Ease of Maintenance:**
- With well-defined roles and responsibilities for objects, behavioral patterns contribute to easier maintenance. Changes to the behavior can be localized, reducing the impact on the rest of the code.

Disadvantages of Behavioral Design Patterns

- **Increased Complexity:** Introducing behavioral patterns can sometimes lead to increased complexity in the codebase, especially when multiple patterns are used or when there is an excessive use of design patterns in general.
- **Over-Engineering:** There is a risk of over-engineering when applying behavioral patterns where simpler solutions would suffice. Overuse of patterns may result in code that is more complex than necessary.
- **Limited Applicability:** Not all behavioral patterns are universally applicable. The suitability of a pattern depends on the specific requirements of the system, and using a pattern in the wrong context may lead to unnecessary complexity.
- **Code Readability:** In certain cases, applying behavioral patterns may make the code less readable and harder to understand, especially for developers who are not familiar with the specific pattern being employed.
- **Scalability Concerns:** As the complexity of a system increases, the scalability of certain behavioral patterns may become a concern. For example, the Observer pattern may become less efficient with a large number of observers.

Downloaded from EnggTree.com

EnggTree.com

Model-view-controller (MVC)

Definition

The MVC pattern in Software Engineering Architecture is defined as an application being separated into three logical components: Model, View and Controller.

Model

This component in the architecture will represent all data-related logic. This includes defining how the data is formed. In other words, this holds the definition for many of the types that we use in the application. In many cases, the model here refers to the type of data that we are dealing with in the application. This component also notifies its dependents about data changes.

www.EnggTree.com

View

Contains all User interface (UI) logic in the application. This component of the application encapsulates mainly the UI related logic which includes things that the end user will manipulate like dropdown buttons and web pages etc.

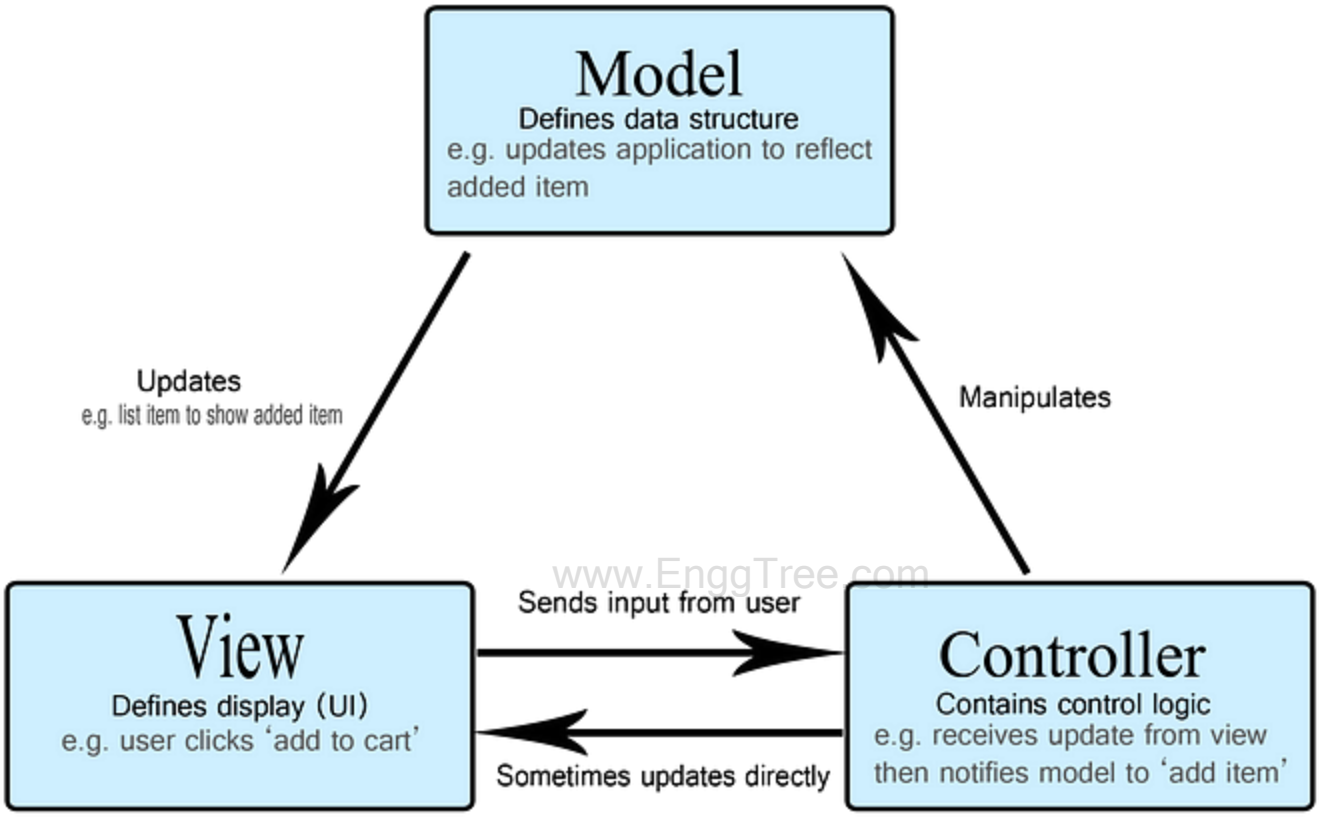
Controller

Controllers exist as a layer between Model and View components to process all the business logic arising from user input. It is responsible to handle inputs from the View

Downloaded from EnggTree.com

EnggTree.com

components, manipulate data using the models from the Model component and then finally interact with the view components again to render the final output to the end user. Responsible for manipulating the data.



Why MVC

The MVC pattern today is widely used for many applications and remain a popular choice. This is due to a few key reasons

Downloaded from EnggTree.com

EnggTree.com

Faster Development Time

Given the separation of the applications into the three distinct areas, this means that more developers are able to work on each part separately. e.g if a developer works on the model, he is not directly blocking another developer from building up the view component of the application and thus allows teams to speed up development purely due to the nature of the architecture itself.

Greater Testability

Each component being separated from each other means that developers are able to test each one separately and in isolation. This is made easier due to the clear separation of concerns that is applied in this architecture pattern. e.g a Model can be tested easily without the view component.

www.EnggTree.com

Easy extension of views/modification

Any changes in view component will usually not affect the model component, hence developers using this pattern can easily extend and add new views to the application to display the data from model in different ways. Thus, modifications are easier to be isolated to a single component instead of affecting the entire application

MVC in Real World application

Web applications

Downloaded from EnggTree.com

EnggTree.com

Many web applications today run primarily on MVC architecture. In particular, ASP.NET MVC framework offers and MVC pattern as one of the development model.

This framework provides developers with a MVC abstraction built on top of ASP.NEXT and thus provide a large set of added functionality.

Sample code from ASP.NET documentation

An example of a controller action in ASP.NET MVC framework.

Another example framework for web that uses MVC is Sails. Sails is a nodeJs framework that provides added functionality. A sails app comes preconfigured with the MVC structure predefined and developers can just use it right out of the box.

www.EnggTree.com

Downloaded from EnggTree.com

EnggTree.com

What is an Adapter?

An adapter is a class that transforms (adapts) an interface into another.

For example, an adapter implements an interface A and gets injected an interface B. When the adapter is instantiated it gets injected in its constructor an object that implements interface B. This adapter is then injected wherever interface A is needed and receives method requests that it transforms and proxies to the inner object that implements interface B.

www.EnggTree.com

If I managed to confuse you, no worries, I give a more concrete example further below. □

Two different types of adapters

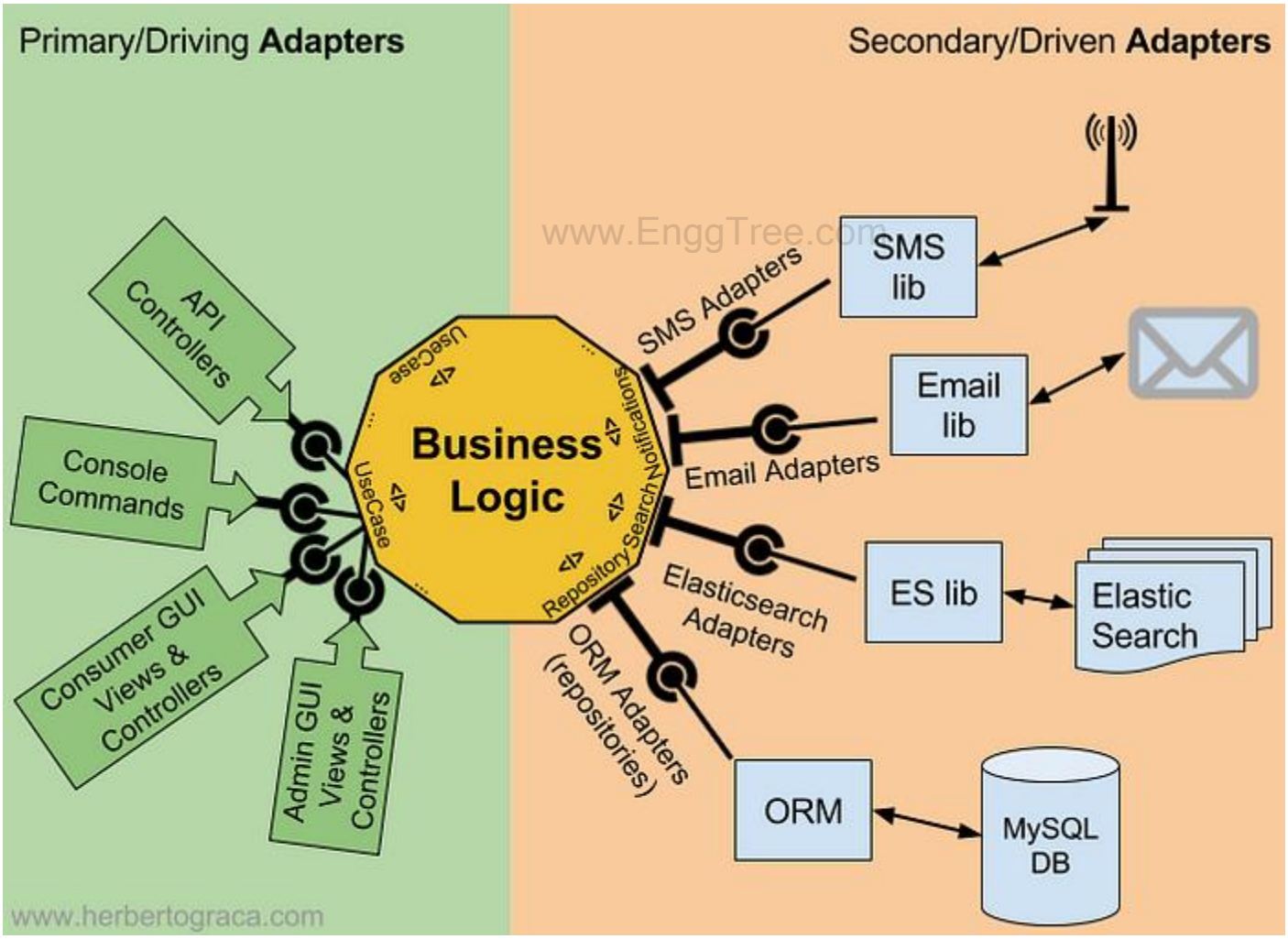
The adapters on the left side, representing the UI, are called the **Primary** or **Driving Adapters** because they are the ones to start some action on the application, while the adapters on the right side, representing the connections to the backend tools, are called the **Secondary** or **Driven Adapters** because they always react to an action of a primary adapter.

There is also a difference on how the ports/adapters are used:

Downloaded from EnggTree.com

EnggTree.com

- On the **left side**, the adapter depends on the port and gets injected a concrete implementation of the port, which contains the use case. On this side, **both the port and its concrete implementation (the use case) belong inside the application**;
- On the **right side**, the adapter **is** the concrete implementation of the port and is injected in our business logic although our business logic only knows about the interface. On this side, **the port belongs inside the application, but its concrete implementation belongs outside** and it wraps around some external tool.



Downloaded from EnggTree.com

EnggTree.com

What are the benefits?

Using this port/adapter design, with our application in the centre of the system, allows us to keep the application isolated from the implementation details like ephemeral technologies, tools and delivery mechanism

COMMAND:

The **command pattern** is a behavioral design pattern in which an object is used to encapsulate all information needed to perform an action or trigger an event at a later time. This information includes the method name, the object that owns the method and values for the method parameters.

Four terms always associated with the command pattern are *command*, *receiver*, *invoker* and *client*. A *command* object knows about *receiver* and invokes a method of the receiver. Values for parameters of the receiver method are stored in the command.

www.EnggTree.com

The receiver object to execute these methods is also stored in the command object by aggregation. The *receiver* then does the work when the `execute()` method in *command* is called. An *invoker* object knows how to execute a command, and optionally does bookkeeping about the command execution.

The invoker does not know anything about a concrete command, it knows only about the command *interface*. Invoker object(s), command objects and receiver objects are held by a *client* object, the *client* decides which receiver objects it assigns to the command objects, and which commands it assigns to the invoker.

The client decides which commands to execute at which points. To execute a command, it passes the command object to the invoker object.

Using command objects makes it easier to construct general components that need to delegate, sequence or execute method calls at a time of their choosing without the need to know the class of the method or the method parameters.

Using an invoker object allows bookkeeping about command executions to be conveniently performed, as well as implementing different modes for commands, which are managed by the invoker object, without the need for the client to be aware of the existence of bookkeeping or modes.

Downloaded from EnggTree.com

EnggTree.com

System Design Strategy – Software Engineering

A good system design is to organize the program modules in such a way that are easy to develop and change. Structured design techniques help developers to deal with the size and complexity of programs. Analysts create instructions for the developers about how code should be written and how pieces of code should fit together to form a program.

Software Engineering is the process of designing, building, testing, and maintaining software. The goal of software engineering is to create software that is reliable, efficient, and easy to maintain. System design is a critical component of software engineering and involves making decisions about the architecture, components, modules, interfaces, and data for a software system.

System Design Strategy refers to the approach that is taken to design a software system. There are several strategies that can be used to design software systems, including the following:

1. Top-Down Design: This strategy starts with a high-level view of the system and gradually breaks it down into smaller, more manageable components.
2. Bottom-Up Design: This strategy starts with individual components and builds the system up, piece by piece.
3. Iterative Design: This strategy involves designing and implementing the system in stages, with each stage building on the results of the previous stage.
4. Incremental Design: This strategy involves designing and implementing a small part of the system at a time, adding more functionality with each iteration.
5. Agile Design: This strategy involves a flexible, iterative approach to design, where requirements and design evolve through collaboration between self-organizing and cross-functional teams.

www.EnggTree.com

The design of a system is essentially a blueprint or a plan for a solution for the system. 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, the specifications of these modules and how the modules should be interconnected. The design of a system is correct if a system built precisely according to the design satisfies the requirements of that system. The goal of the design process is not simply to produce a design for the system. Instead, the goal is to find the best possible design within the limitations imposed by the requirements and the physical and social environment in which the system will operate.

The choice of system design strategy will depend on the particular requirements of the software system, the size and complexity of the system, and the development methodology being used. A well-designed system can simplify the development process, improve the quality of the software, and make the software easier to maintain.

Importance of System Design Strategy:

1. If any pre-existing code needs to be understood, organized, and pieced together.
2. It is common for the project team to have to write some code and produce original programs that support the application logic of the system.

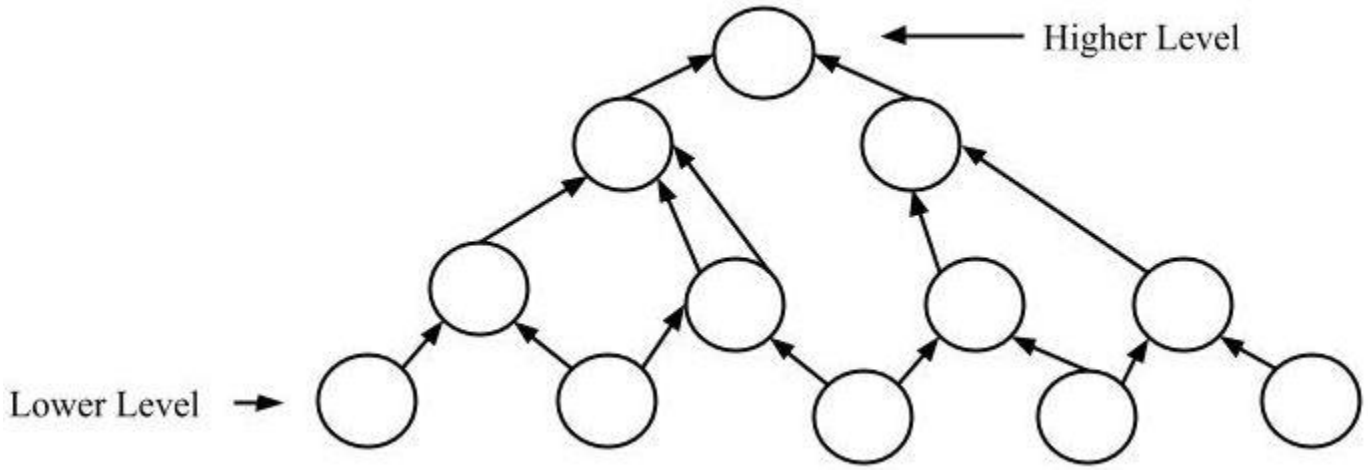
There are many strategies or techniques for performing system design. They are:

Bottom-up approach:

Downloaded from EnggTree.com

EnggTree.com

The design starts with the lowest level components and subsystems. By using these components, the next immediate higher-level components and subsystems are created or composed. The process is continued till all the components and subsystems are composed into a single component, which is considered as the complete system. The amount of abstraction grows high as the design moves to more high levels.
By using the basic information existing system, when a new system needs to be created, the bottom-up strategy suits the purpose.



Bottom-up approach

www.EnggTree.com

Advantages of Bottom-up approach:

- The economics can result when general solutions can be reused.
- It can be used to hide the low-level details of implementation and be merged with the top-down technique.

Disadvantages of Bottom-up approach:

- It is not so closely related to the structure of the problem.
- High-quality bottom-up solutions are very hard to construct.
- It leads to the proliferation of ‘potentially useful’ functions rather than the most appropriate ones.

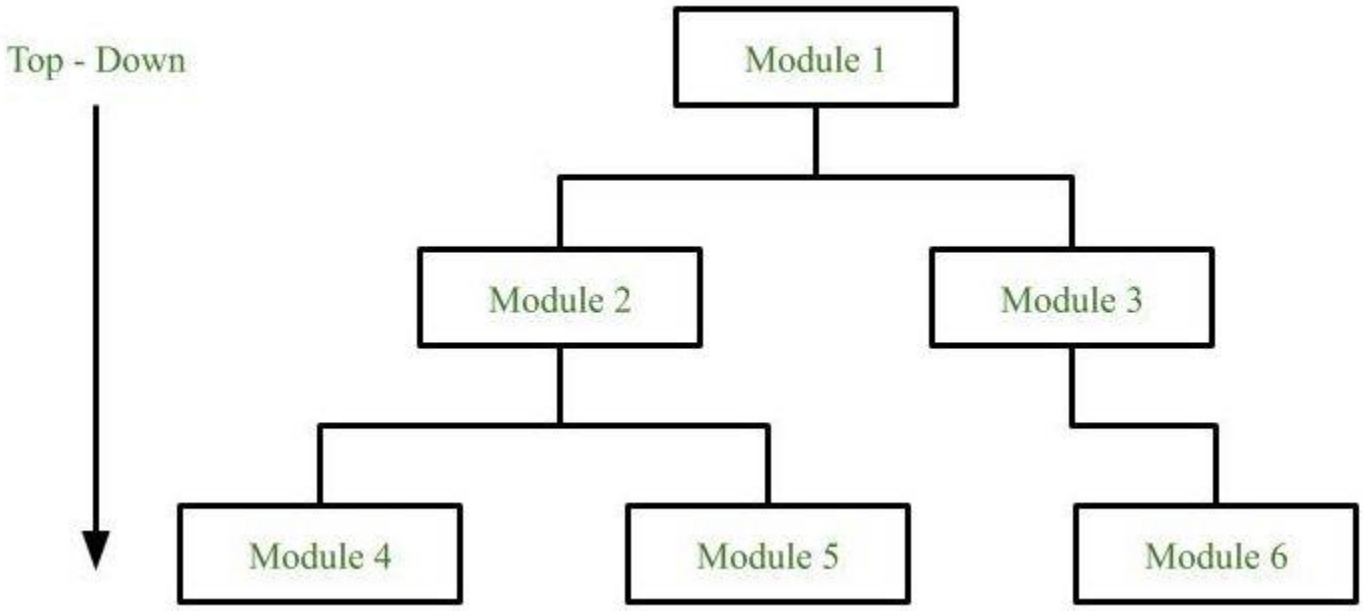
Top-down approach:

Each system is divided into several subsystems and components. Each of the subsystems is further divided into a set of subsystems and components. This process of division facilitates forming a system hierarchy structure. The complete software system is considered a single entity and in relation to the characteristics, the system is split into sub-systems and components. The same is done with each of the sub-systems.
This process is continued until the lowest level of the system is reached. The design is started initially by defining the system as a whole and then keeps on adding definitions of the subsystems and components. When all the definitions are combined, it turns out to be a complete system.

Downloaded from EnggTree.com

EnggTree.com

For the solutions of the software that need to be developed from the ground level, a top-down design best suits the purpose.



Top-down approach

Advantages of Top-down approach:

www.EnggTree.com

- The main advantage of the top-down approach is that its strong focus on requirements helps to make a design responsive according to its requirements.

Disadvantages of Top-down approach:

- Project and system boundaries tend to be application specification-oriented. Thus, it is more likely that the advantages of component reuse will be missed.
- The system is likely to miss, the benefits of a well-structured, simple architecture.
- **Hybrid Design:**
  It is a combination of both top-down and bottom-up design strategies. In this, we can reuse the modules.

Advantages of using a System Design Strategy:

1. Improved quality: A well-designed system can improve the overall quality of the software, as it provides a clear and organized structure for the software.
2. Ease of maintenance: A well-designed system can make it easier to maintain and update the software, as the design provides a clear and organized structure for the software.
3. Improved efficiency: A well-designed system can make the software more efficient, as it provides a clear and organized structure for the software that reduces the complexity of the code.
4. Better communication: A well-designed system can improve communication between stakeholders, as it provides a clear and organized structure for the software that makes it easier for stakeholders to understand and agree on the design of the software.

Downloaded from EnggTree.com

5. Faster development: A well-designed system can speed up the development process, as it provides a clear and organized structure for the software that makes it easier for developers to understand the requirements and implement the software.

Disadvantages of using a System Design Strategy:

1. Time-consuming: Designing a system can be time-consuming, especially for large and complex systems, as it requires a significant amount of documentation and analysis.
2. Inflexibility: Once a system has been designed, it can be difficult to make changes to the design, as the process is often highly structured and documentation-intensive.

PUBLISH –SUBSCRIBE

The publisher-subscriber (pub-sub) model is a widely used architectural pattern. We can use it in software development to enable communication between different components in a system.

In particular, it is often used in distributed systems, where different parts of the system need to interact with each other but don't want to be tightly coupled.

In this tutorial, we'll explore the pub-sub model, how it works, and some common use cases for this architectural pattern.

www.EnggTree.com

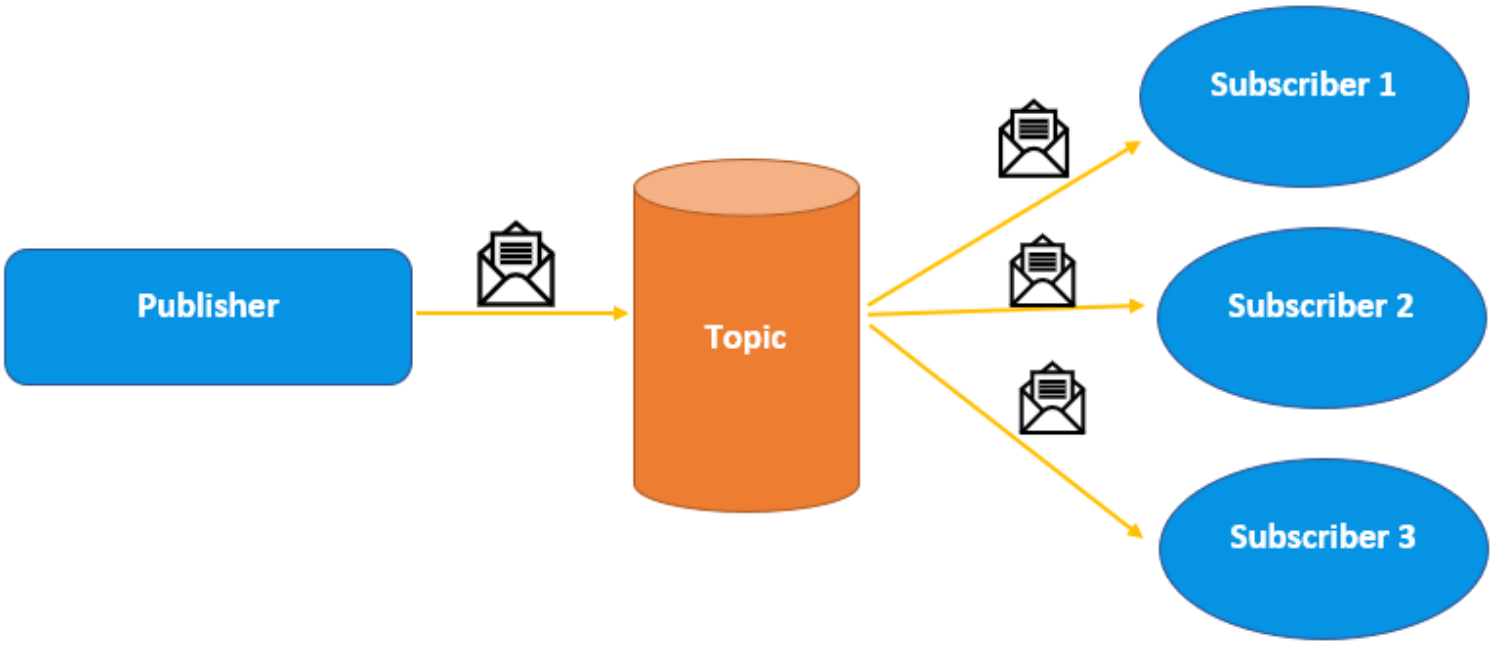
2. Pub-Sub Model: Overview

The pub-sub model involves publishers and subscribers, making it a messaging pattern. Specifically, the publishers are responsible for sending messages to the system, while subscribers are responsible for receiving those messages.

Mainly, the pub-sub model is based on decoupling components in a system, which means that components can interact without being tightly coupled.

Downloaded from EnggTree.com

EnggTree.com



3. How the Pub-Sub Model Works

In this section, we'll discuss how this model works, including sending messages, checking for subscribers, receiving messages, registering for topics, decoupling publishers and subscribers, and additional features the message broker implements to enhance message delivery.

3.1. Sending Messages

A publisher sends a message to the message broker with a specific topic, which is a string that identifies the content of the message.

3.2. Checking for Subscribers

The message broker receives the message and checks the topic to see if any subscribers have expressed interest in receiving messages on that topic. Furthermore, if subscribers are interested in the topic, the message broker sends the message to all subscribers who have registered interest in that topic.

Downloaded from EnggTree.com

EnggTree.com

3.3. Receiving Messages

Subscribers receive the message from the message broker. Then, it can process the message as needed. However, the message is discarded if no subscribers are interested in the topic.

3.4. Registering for Topics

To receive messages on specific topics, subscribers can register interest in one or more topics with the message broker. Additionally, this feature enables subscribers to receive messages on topics they are interested in.

3.5. Decoupling Publishers and Subscribers

Publishers and subscribers do not need to know about each other's existence since they interact only through the message broker, which acts as an intermediary.

3.6. Additional Features

www.EnggTree.com

The message broker can also implement additional features such as filtering messages based on content, ensuring message delivery, and providing message ordering guarantees. These features enhance the reliability and efficiency of message delivery.

By decoupling publishers and subscribers, the pub-sub model allows them to interact through a message broker, which helps to reduce tight coupling between components in a system.

This makes it an ideal messaging pattern for use in distributed systems, where different parts of the system must interact without being tightly coupled.

4. Advantages and Disadvantages of the Pub-Sub Model

The pub-sub model has several benefits. The following table summarizes its main advantages:

Downloaded from EnggTree.com

EnggTree.com

The pub-sub model has several benefits. The following table summarizes its main advantages:

Advantage	Description
Scalability	The decoupled nature of the pub-sub model makes it highly scalable. The model can handle a large number of publishers and subscribers without affecting the performance
Reliability	A message broker ensures the reliable delivery of messages to interested subscribers, even if some subscribers are offline or disconnected
Flexibility	The pub-sub model offers high flexibility by enabling the addition or removal of publishers and subscribers without affecting the overall system
Loose coupling	The decoupled nature of the pub-sub model ensures that publishers and subscribers are loosely coupled, which allows them to evolve independently without affecting each other

However, the pub-sub model also has some drawbacks. The following table shows its main drawbacks:

www.EnggTree.com

Disadvantage	Description
Increased complexity	The use of a message broker adds complexity to the system, making it more difficult to implement and maintain
Higher latency	The use of a message broker can introduce additional latency into the system, which may be unacceptable for some real-time applications
Single point of failure	The message broker represents a single point of failure for the system, which may result in service disruption if it fails
Loose coupling	The decoupled nature of the pub-sub model ensures that publishers and subscribers are loosely coupled, allowing them to evolve independently without affecting each other

5. Use Cases for the Pub-Sub Model

Downloaded from EnggTree.com

EnggTree.com

In this section, we'll explore some use cases of this model, including **real-time updates in online games**, **smart homes with IoT**, and **data distribution in data analytics.**

5.1. Real-time Updates in Online Games

One of the use cases for the pub-sub model is online gaming, where publishers can send real-time updates on player positions, score changes, and game events to all subscribers.

Overall, this enhances the gaming experience and ensures all players receive the same updates simultaneously.

5.2. Smart Homes with IoT

The pub-sub model is also used in smart homes to send messages from sensors to actuators. For example, we can use this model to turn on the lights when someone enters a room.

www.EnggTree.com

Downloaded from EnggTree.com

EnggTree.com

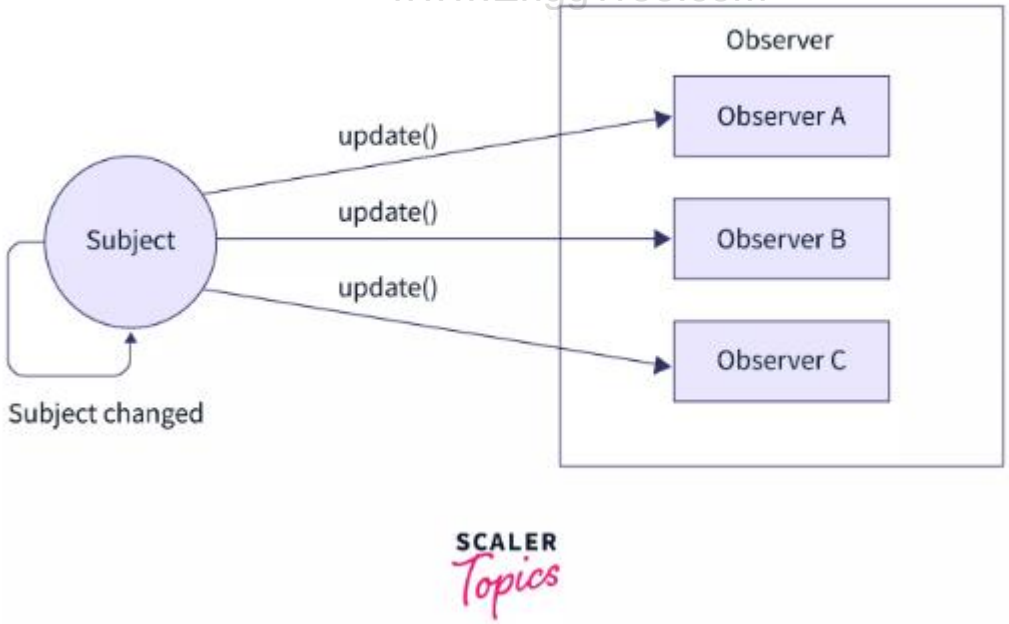
OBSERVER

Observer design pattern falls under the category of behavioral design patterns. The Observer Pattern maintains a one-to-many relationship among objects, ensuring that when the state of one object is changed, all of its dependent objects are simultaneously informed and updated. This design pattern is also referred to as Dependents.

A subject and observer (many) exist in a one-to-many relationship. Here the observers do not have any access to data, so they are dependent on the subject to feed them with information.

The observer design patterns when designing a system where several objects are interested in any possible modification to a specific object. In other words, the observer design pattern is employed when there is a one-to-many relationship between objects, such as when one object is updated, its dependent objects must be automatically notified.

www.EnggTree.com



SCALER Topics

Real-World Example: If a bus gets delayed, then all the passengers who were supposed to travel in it get notified about the delay, to minimize inconvenience. Here, the bus agency is the subject and all the passengers are the observers. All the passengers are dependent on the agency to provide them with information about

Downloaded from EnggTree.com

EnggTree.com

any changes regarding their travel journey. Hence, there is a one-to-many relationship between the travel agency and the passengers.

Other Examples of observer design patterns:

- **Social Media Platforms for example, many users can follow a particular person(subject) on a social media platform. All the followers will be updated if the subject updates his/her profile. The users can follow and unfollow the subject anytime they want.**
- **Newsletters or magazine subscription**
- **Journalists providing news to the media**
- **E-commerce websites notify customers of the availability of specific products**

Consider the following scenario: There is a new laughter club in town, with a grand opening, it caught the attention of a lot of people interested in being a member of the club. Thrilled with the overwhelming response, the club owner was a bit worried about the smooth management and involvement of all the members.

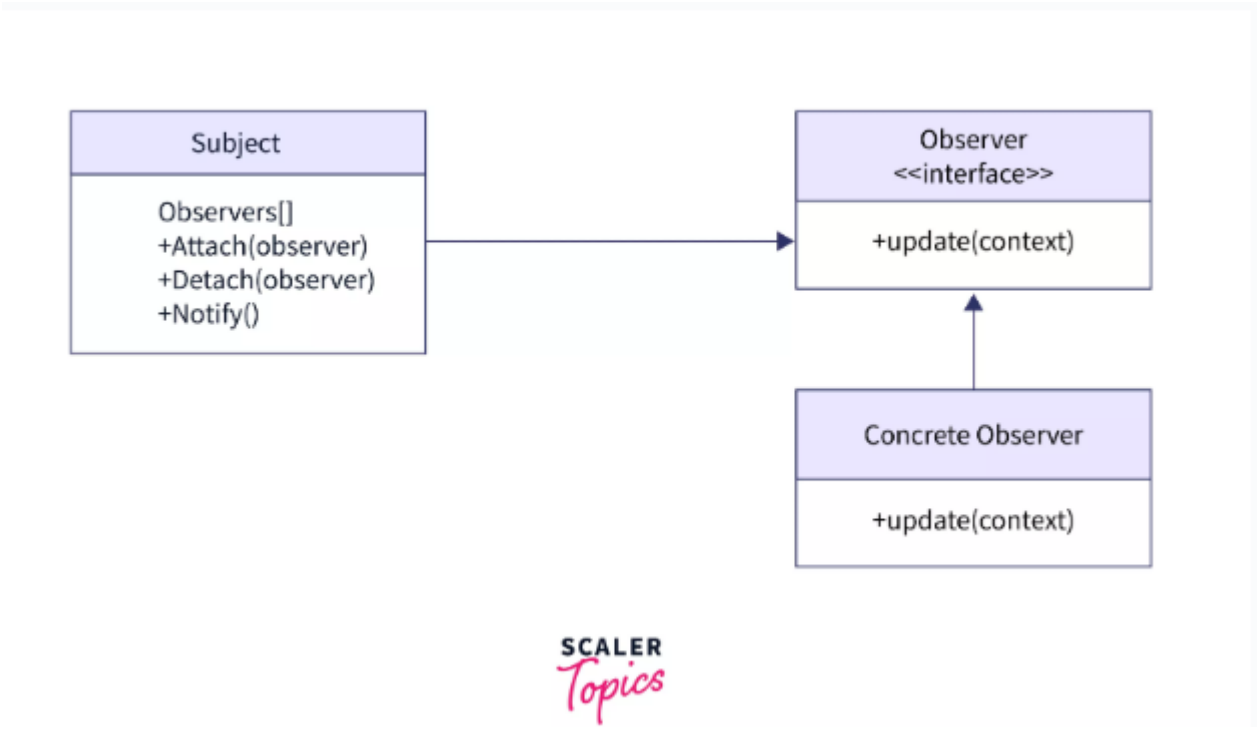
The observer design pattern is the best solution to the owner's problem. The owner here is the subject and all the members of the club are observers. The observers have no access to the club's information and the upcoming events unless the owner notifies them of it. Also, the members have the option to opt out of the club whenever they want to. This allows the owner to easily manage and engage all of the members.

www.EnggTree.com

Downloaded from EnggTree.com

EnggTree.com

How does Observer Design Patterns work?



SCALER Topics

Structure

www.EnggTree.com

- **The subject delivers events that are intriguing to the observers. These events occur due to changes in the state of the subject or the execution of certain behaviors. Subjects have a registration architecture that enables new observers to join and existing observers to withdraw from the list.**
- **Whenever a new event occurs, the subject iterates through the list of observers, calling the notify method provided in the 'observer' interface.**
- **The notification interface is declared by the Observer interface. It usually includes an updating method. The method may include numerous options that allow the subject to provide event information together with the update.**
- **Concrete Observers do certain activities in response to alerts sent by the Subject. All the concrete observer classes should implement the base observer interface, and the subject interface is coupled only with the base observer interface.**
- **Sometimes, observers want some additional context in order to properly process any update notified by the Subject. As a result, the Subject frequently supplies some contextual information as parameters to the notification function. The subject can pass itself as an argument, allowing the subject to immediately retrieve any needed data.**

Downloaded from EnggTree.com

EnggTree.com

Implementation

1. Declare an Observer interface with an update method at the least.
2. Declare the subject interface and a couple of methods for attaching and detaching observer objects from the list of observers. (Remember that publishers must only interact with subscribers through the subscriber interface.)
3. Create a concrete subject class if needed and add the subscribers' list to this class. Since this concrete class extends the Subject interface, it inherits all its properties including adding and removing observers. Every time something significant happens inside the subject class, it must inform to all the observers about it.
4. In concrete observer classes, add the update notification methods. Some observers might want basic background information about the occurrence. The notification method accepts it as an input.

PROXY DESIGN PATTERN

Proxy means 'in place of', representing' or 'in place of' or 'on behalf of' are literal meanings of proxy and that directly explains Proxy Design Pattern. Proxies are also called surrogates, handles, and wrappers. They are closely related in structure, but not purpose, to Adapters and Decorators.

www.EnggTree.com

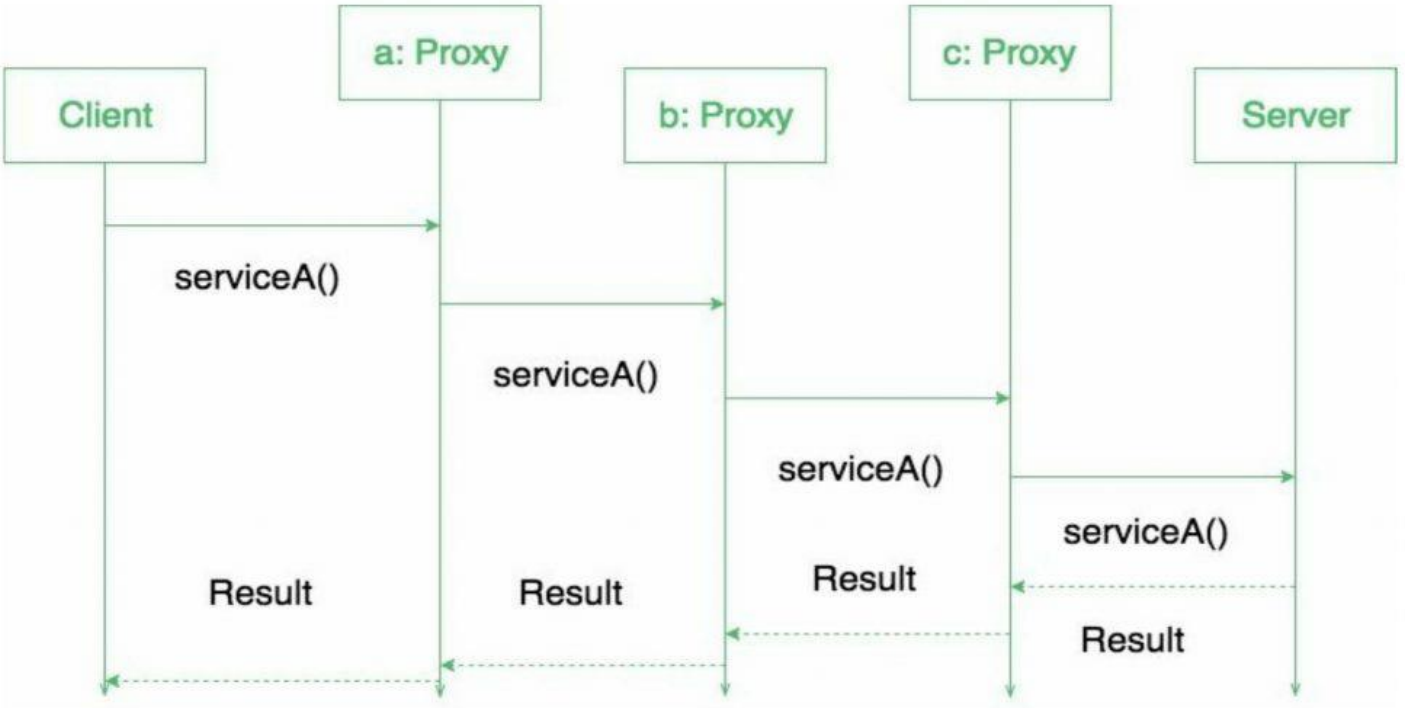
A real world example can be a cheque or credit card is a proxy for what is in our bank account. It can be used in place of cash, and provides a means of accessing that cash when required. And that's exactly what the Proxy pattern does – "Controls and manage access to the object they are protecting".

BEHAVIOR

As in the decorator pattern, proxies can be chained together. The client, and each proxy, believes it is delegating messages to the real server:

Downloaded from EnggTree.com

EnggTree.com



www.EnggTree.com

When to use this pattern?

Proxy pattern is used when we need to create a wrapper to cover the main object's complexity from the client.

TYPES OF PROXIES

Remoteproxy:

They are responsible for representing the object located remotely. Talking to the real object might involve marshalling and unmarshalling of data and talking to the remote object. All that logic is encapsulated in these proxies and the client application need not worry about them.

Virtual proxy:

These proxies will provide some default and instant results if the real object is supposed to take some time to produce results. These proxies initiate the operation on real objects and provide a default result to the application. Once the real object is done, these proxies push the actual data to the client where it has provided dummy data earlier.

Protection proxy:

Downloaded from EnggTree.com

EnggTree.com

If an application does not have access to some resource then such proxies will talk to the objects in applications that have access to that resource and then get the result back.

Smart Proxy:

A smart proxy provides additional layer of security by interposing specific actions when the object is accessed. An example can be to check if the real object is locked before it is accessed to ensure that no other object can change it.

Some Examples

A very simple real life scenario is our college internet, which restricts few site access. The proxy first checks the host you are connecting to, if it is not part of restricted site list, then it connects to the real internet. This example is based on Protection proxies.

Interface of Internet

```
package com.saket.demo.proxy;

public interface Internet
```

www.EnggTree.com

```
{

    public void connectTo(String serverhost) throws Exception;

}
```

Benefits:

- **One of the advantages of Proxy pattern is security.**
- **This pattern avoids duplication of objects which might be huge size and memory intensive. This in turn increases the performance of the application.**
- **The remote proxy also ensures about security by installing the local code proxy (stub) in the client machine and then accessing the server with help of the remote code.**

Drawbacks/Consequences:

This pattern introduces another layer of abstraction which sometimes may be an issue if the RealSubject code is accessed by some of the clients directly and some of them might access the Proxy classes. This might cause disparate behaviour.

Downloaded from EnggTree.com

EnggTree.com

FAÇADE:

The facade pattern (also spelled façade) is a software-design pattern commonly used in object-oriented programming. Analogous to a facade in architecture, a facade is an object that serves as a front-facing interface masking more complex underlying or structural code.

Facade is a part of the Gang of Four design patterns and it is categorized under Structural design patterns. Before we dig into the details of it, let us discuss some examples which will be solved by this particular Pattern. So, As the name suggests, it means the face of the building.

The people walking past the road can only see the glass face of the building. They do not know anything about it, the wiring, the pipes, and other complexities. It hides all the complexities of the building and displays a friendly face.

More examples

www.EnggTree.com

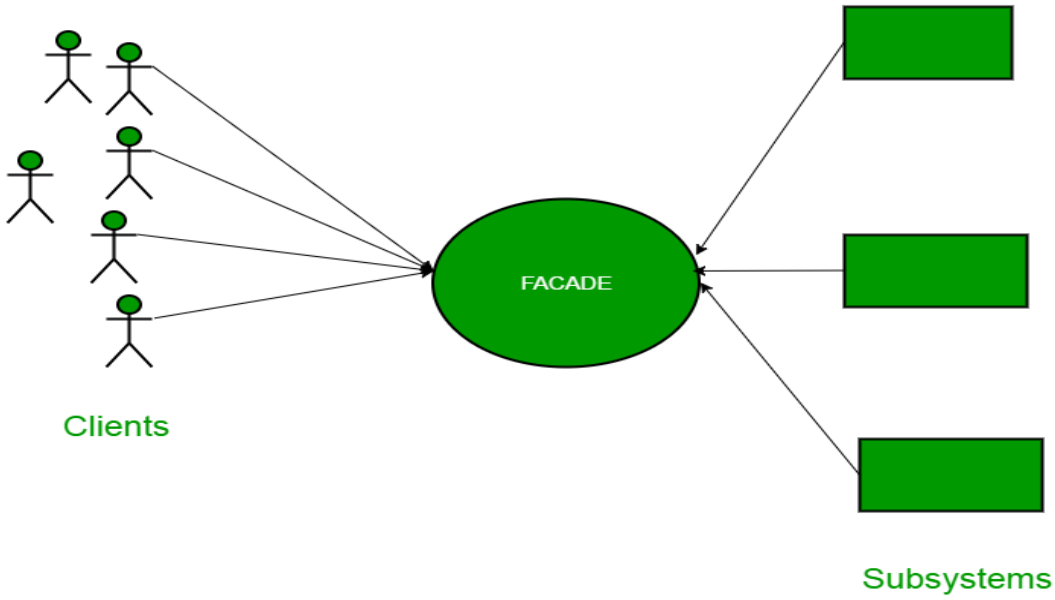
In Java, the interface JDBC can be called a facade because we as users or clients create connections using the "java.sql.Connection" interface, the implementation of which we are not concerned about.

The implementation is left to the vendor of the driver. Another good example can be the startup of a computer. When a computer starts up, it involves the work of CPU, memory, hard drive, etc.

To make it easy to use for users, we can add a facade that wraps the complexity of the task, and provide one simple interface instead. The same goes for the Facade Design Pattern. It hides the complexities of the system and provides an interface to the client from where the client can access the system.

Downloaded from EnggTree.com

EnggTree.com



FACADE DESIGN PATTERN DIAGRAM

Now Let's try and understand the facade pattern better using a simple example. Let's consider a hotel. This hotel has a hotel keeper.

There are a lot of restaurants inside the hotel e.g. Veg restaurants, Non-Veg restaurants, and Veg/Non Both restaurants. You, as a client want access to different menus of different restaurants. You do not know what are the different menus they have. You just have access to a hotel keeper who knows his hotel well. Whichever menu you want, you tell the hotel keeper and he takes it out of the respective restaurants and hands it over to you. Here, the hotel keeper acts as the facade, as he hides the complexities of the system hotel. Let's see how it works:

1. **Facade is a structural design pattern with the intent to provide a simplified (but limited) interface to a complex system of classes, library or framework.**
2. **A Facade class can often be transformed into a Singleton since a single facade object is sufficient in most cases.**
3. **Facade routinely wraps multiple objects.**

Downloaded from EnggTree.com

EnggTree.com

4. **The Facade class will not encapsulate subclasses but will provide a simple interface to their functions.**

5. **Classes in the subsystem will still be available to the client. However, Facade will decouple client and subsystems so that they can change independently.**

6. **We can combine different features by writing different Facade classes for different clients.**

7. **Facade can be recognized in a class that has a simple interface, but delegates most of the work to other classes.**

8. **Usually, facades manage the full life cycle of objects they use.**

9. **Client is shielded from the unwanted complexities of the subsystems and gets only to a fraction of a subsystem's capabilities.**

www.EnggTree.com

10. **Abstract Factory can serve as an alternative to Facade when you only want to hide the way the subsystem objects are created from the client code.**

Usage:

1. **Use the Facade pattern when you need to have a limited but straightforward interface to a complex subsystem.**

2. **Use the Facade when you want to structure a subsystem into layers.**

Related Patterns:

1. **Combined with Abstract Factory or used as an alternative to it.**

2. **Often confused with Adapter and mediator patterns.**

Downloaded from EnggTree.com

EnggTree.com

Some use cases in Enterprise Applications:

1. **Decoupling Application Code From the Library.**
2. **Reusing Legacy Code in New Application.**
3. **Fixing Interface Segregation Principle Violation.**

www.EnggTree.com

Downloaded from EnggTree.com

EnggTree.com

ARCHITECTURAL STYLES

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 centered architectures:

www.EnggTree.com

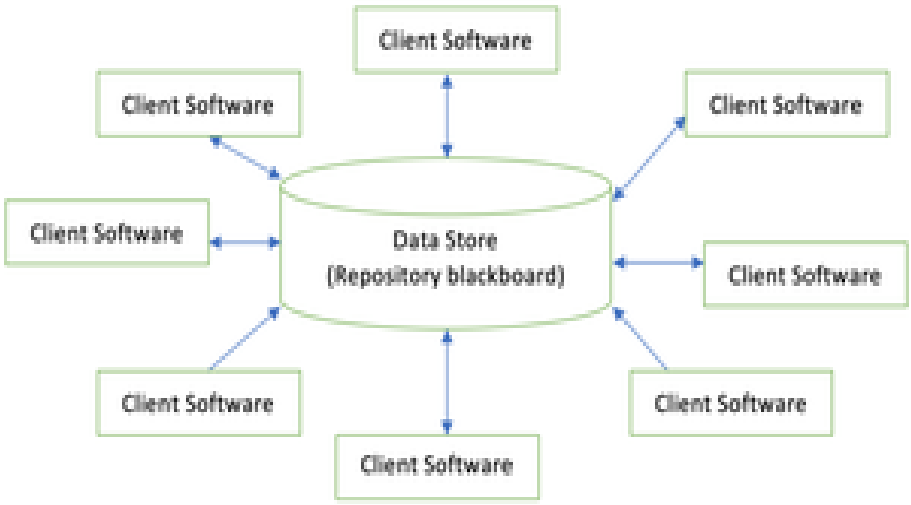
- A data store will reside at the center of this architecture and is accessed frequently by the other components that update, add, delete or modify the data present within the store.
- The figure illustrates a typical data centered style. The client software access a central repository. Variation of 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.

Advantage of Data centered architecture

- Repository of data is independent of clients
- Client work independent of each other
- It may be simple to add additional clients.
- Modification can be very easy

Downloaded from EnggTree.com

EnggTree.com



Data centered architecture

2] DATA FLOW ARCHITECTURES:

- This kind of architecture is used when input data is transformed into output data through a series of computational manipulative components.
- The figure represents pipe-and-filter architecture since it uses both pipe and filter and it has a set of components called filters connected by lines.
- Pipes are used to transmitting data from one component to the next.
- Each filter will work independently and is designed to take data input of a certain form and produces data output to the next filter of a specified form. The filters don't require any knowledge of the working of neighboring filters.
- If the data flow degenerates into a single line of transforms, then it is termed as batch sequential. This structure accepts the batch of data and then applies a series of sequential components to transform it.

Advantages of Data Flow architecture

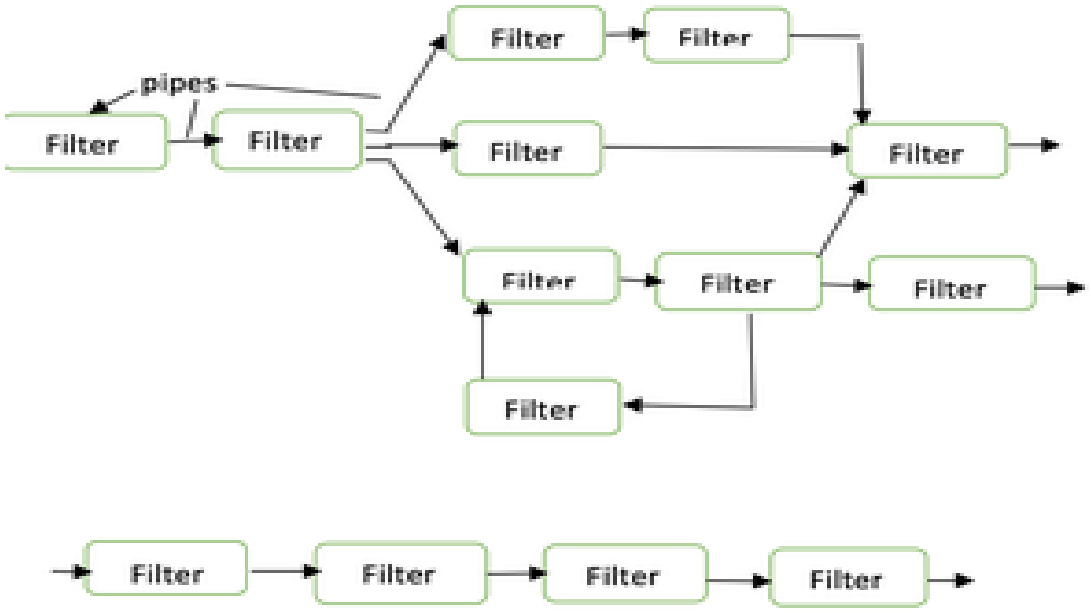
- It encourages upkeep, repurposing, and modification.
- With this design, concurrent execution is supported.

The disadvantage of Data Flow architecture

- It frequently degenerates to batch sequential system
- Data flow architecture does not allow applications that require greater user engagement.
- It is not easy to coordinate two different but related streams

Downloaded from EnggTree.com

EnggTree.com

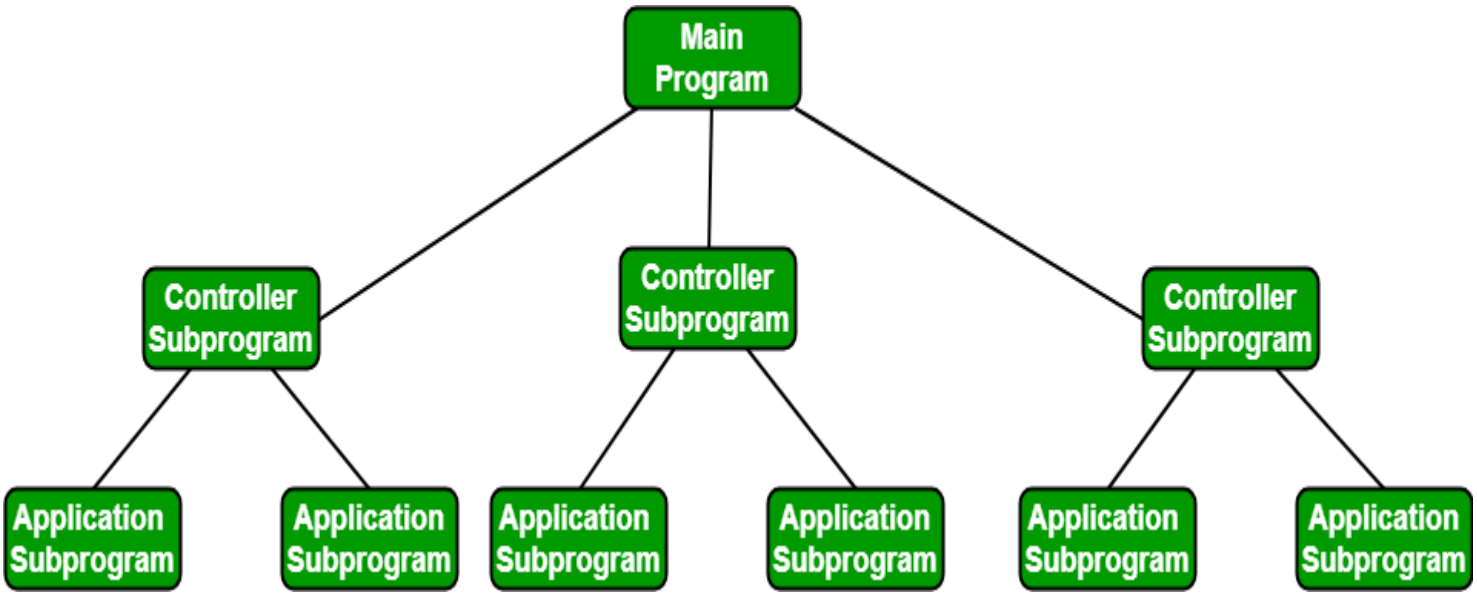


Data Flow architecture

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

- **Remote procedure call architecture:** This components is used to present in a main program or sub program architecture distributed among multiple computers on a network.
- **Main program or Subprogram architectures:** The main program structure decomposes into number of subprograms or function into a control hierarchy. Main program contains number of subprograms that can invoke other components.

www.EnggTree.com



Downloaded from EnggTree.com

EnggTree.com

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.

Characteristics of Object Oriented architecture

- Object protect the system's integrity.
- An object is unaware of the depiction of other items.

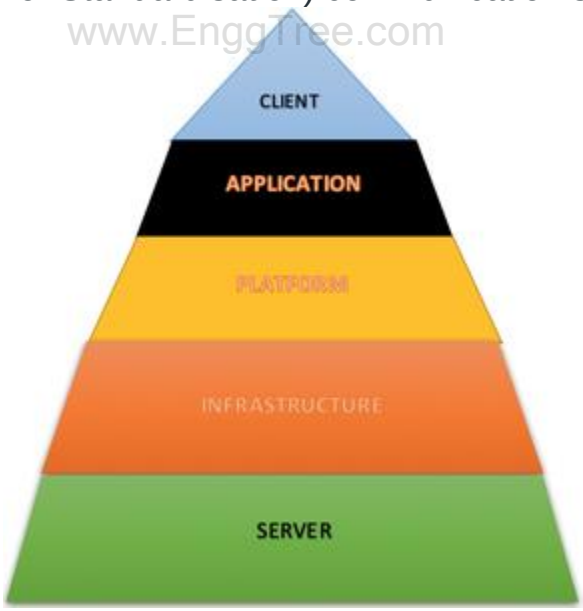
Advantage of Object Oriented architecture

- It enables the designer to separate a challenge into a collection of autonomous objects.
- Other objects are aware of the implementation details of the object, allowing changes to be made without having an impact on other objects.

5] Layered architecture:

- A number of different layers are defined with each layer performing a well-defined set of operations. Each layer will do some operations that becomes closer to machine instruction set progressively.
- At the outer layer, components will receive the user interface operations and at the inner layers, components will perform the operating system interfacing(communication and coordination with OS)
- Intermediate layers to utility services and application software functions.
- One common example of this architectural style is OSI-ISO (Open Systems Interconnection-International Organisation for Standardisation) communication system.

www.EnggTree.com



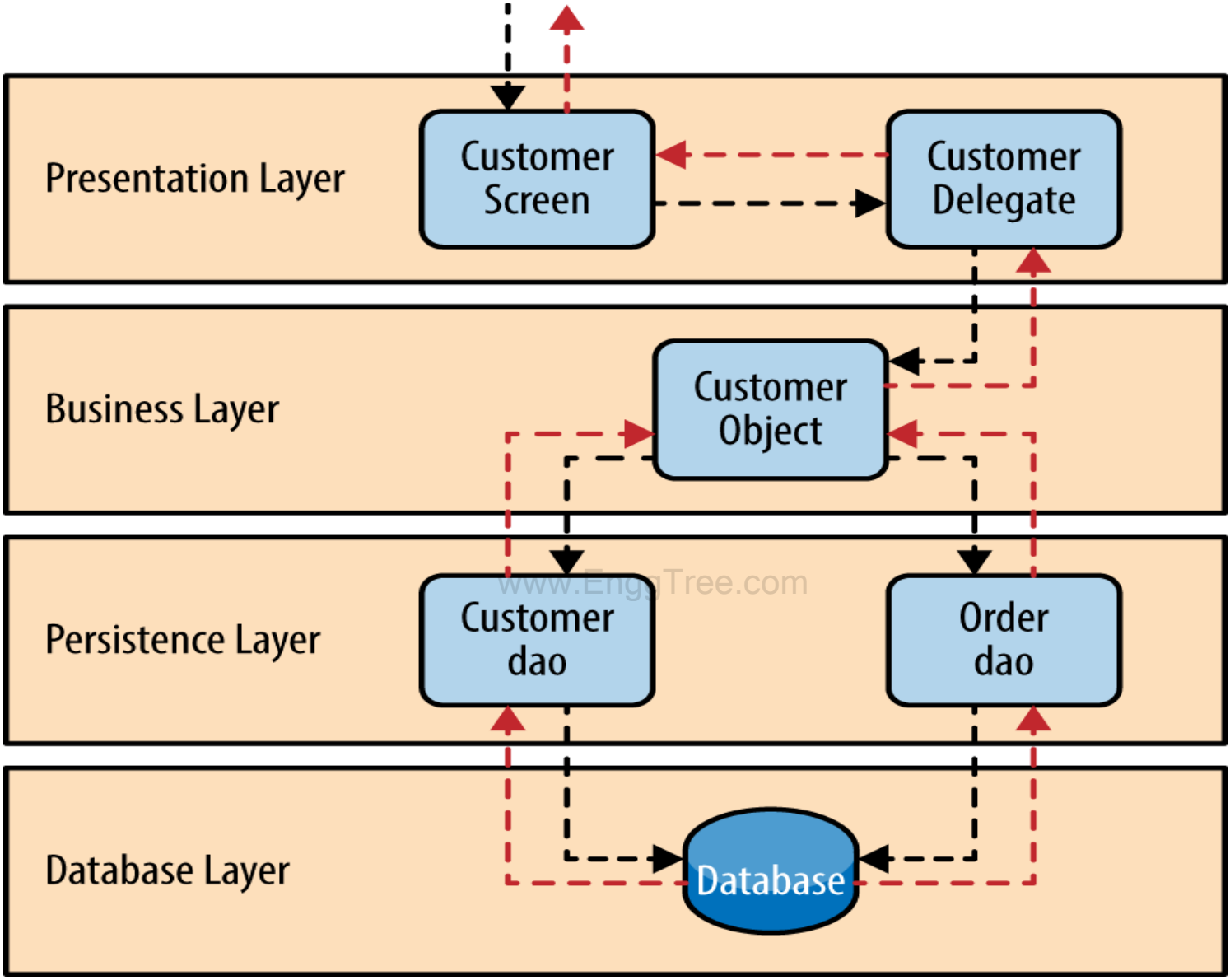
Layered architecture:

The most common architecture pattern is the layered architecture pattern, otherwise known as the n-tier architecture pattern. This pattern is the de facto standard for most Java EE applications and therefore is widely known by most architects, designers, and developers.

Downloaded from EnggTree.com

EnggTree.com

The layered architecture pattern closely matches the traditional IT communication and organizational structures found in most companies, making it a natural choice for most business application development efforts.



Layered architectures are widely used in software development and are considered to be the most prevalent and commonly **utilized architectural framework**.

Downloaded from EnggTree.com

EnggTree.com

It is an **architectural pattern** made up of numerous **independent horizontal layers** that work together to form a **single software unit**. This pattern is also known as n-tier architecture.

A layer is a logical division of parts or programme code. This architecture typically places related or comparable components on the same tiers. Every layer is unique and affects a different aspect of the entire system.

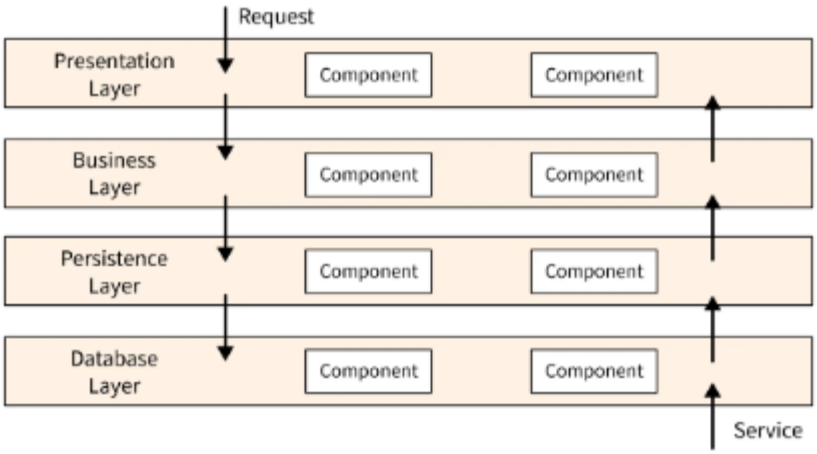
Pattern Description

- Within the application, each layer of the **layered architecture pattern** is responsible for a particular task.
- For instance, the user interface and browser communication logic would be handled by the presentation layer, whilst the **business layer** would be in charge of carrying out the specific business rules related to the request.
- Each layer of the architecture creates an **abstraction** around the work required to complete a specific business requirement.
- For instance, the presentation layer just needs to display customer data on a screen in a specific way; it is not required to understand or worry about how to obtain **customer data**.
- The business layer only needs to obtain the data from the **persistence layer, apply business logic to the data** (e.g., calculate values or aggregate data), and then pass that information up to the presentation layer.
- In a similar manner, the business layer need not worry about how to format customer data for **display on a screen** or even where the customer data is coming from.
-

www.EnggTree.com

Downloaded from EnggTree.com

EnggTree.com



- **Separating** concerns among components is one of the layered architecture pattern's strong points. Components inside that layer deal with only logic specific to a certain layer.
- For instance, the presentation layer's components only deal with **presentation logic**, whereas the business layer's components only deal with business logic. Due to the clearly defined component interfaces and constrained component scope, this type of component classification makes it simple to **incorporate efficient roles** and responsibility models into your architecture. It also makes it simple to develop, test, govern, and maintain applications using this **architecture pattern**.

www.EnggTree.com

CLIENT-SERVER MODEL

The Client-server model is a distributed application structure that partitions task or workload between the providers of a resource or service, called servers, and service requesters called clients. In the client-server architecture, when the client computer sends a request for data to the server through the internet, the server accepts the requested process and deliver the data packets requested back to the client.

How the Client-Server Model works ?

In this article we are going to take a dive into the **Client-Server** model and have a look at how the **Internet** works via, web browsers. This article will help us in having a solid foundation of the WEB and help in working with WEB technologies with ease.

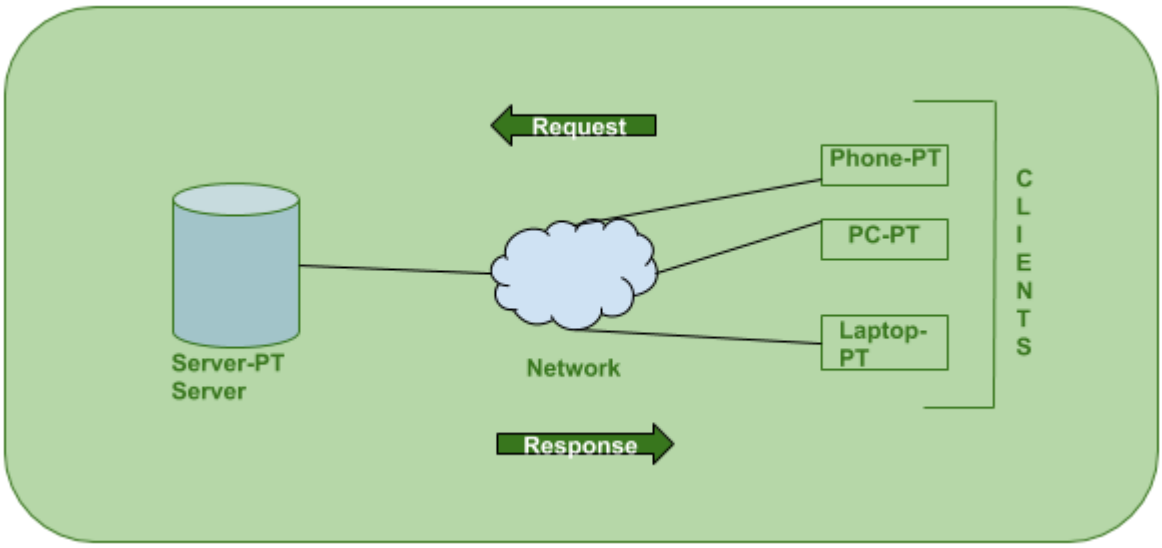
- **Client:** When we talk the word **Client**, it mean to talk of a person or an organization using a particular service. Similarly in the digital world a **Client** is a computer (**Host**) i.e. capable of receiving information or using a particular service from the service providers (**Servers**).

Downloaded from EnggTree.com

EnggTree.com

- **Servers:** Similarly, when we talk the word **Servers**, It mean a person or medium that serves something. Similarly in this digital world a **Server** is a remote computer which provides information (data) or access to particular services.

So, its basically the **Client** requesting something and the **Server** serving it as long as its present in the database.



How the browser interacts with the servers ?

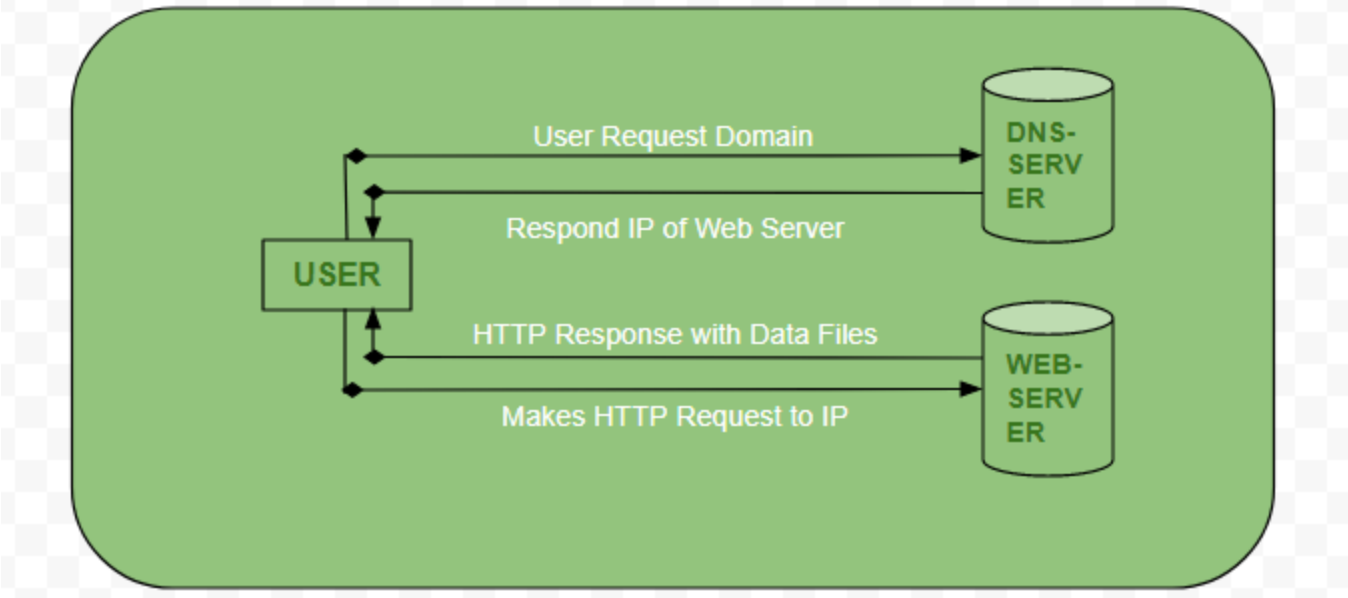
www.EnggTree.com

There are few steps to follow to interacts with the servers a client.

- User enters the **URL**(Uniform Resource Locator) of the website or file. The Browser then requests the **DNS**(DOMAIN NAME SYSTEM) Server.
- **DNS Server** lookup for the address of the **WEB Server**.
- **DNS Server** responds with the **IP address** of the **WEB Server**.
- Browser sends over an **HTTP/HTTPS** request to **WEB Server's IP** (provided by **DNS server**).
- Server sends over the necessary files of the website.
- Browser then renders the files and the website is displayed. This rendering is done with the help of **DOM** (Document Object Model) interpreter, **CSS** interpreter and **JS Engine** collectively known as the **JIT** or (Just in Time) Compilers.

Downloaded from EnggTree.com

EnggTree.com



Advantages of Client-Server model:

- Centralized system with all data in a single place.
- Cost efficient requires less maintenance cost and Data recovery is possible.
- The capacity of the Client and Servers can be changed separately.

Disadvantages of Client-Server model:

- Clients are prone to viruses, Trojans and worms if present in the Server or uploaded into the Server.
- Server are prone to Denial of Service (DOS) attacks.
- Data packets may be spoofed or modified during transmission.
- Phishing or capturing login credentials or other useful information of the user are common and MITM(Man in the Middle) attacks are common.

www.EnggTree.com

Downloaded from EnggTree.com

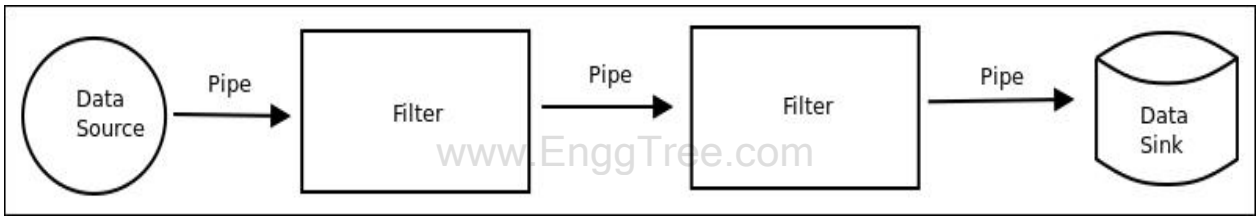
EnggTree.com

PIPE AND FILTER ARCHITECTURES

Pipe and Filter is a simple architectural style that connects a number of components that process a stream of data, each connected to the next component in the processing pipeline via a **Pipe**.

The Pipe and Filter architecture is inspired by the Unix technique of connecting the output of an application to the input of another via pipes on the shell.

The pipe and filter architecture consists of one or more data sources. The data source is connected to data filters via pipes. Filters process the data they receive, passing them to other filters in the pipeline. The final data is received at a **Data Sink**:



Definition

Pipe and Filter is another architectural pattern, which has independent entities called **filters** (components) which perform transformations on data and process the input they receive, and **pipes**, which serve as connectors for the stream of data being transformed, each connected to the next component in the pipeline.

Downloaded from EnggTree.com

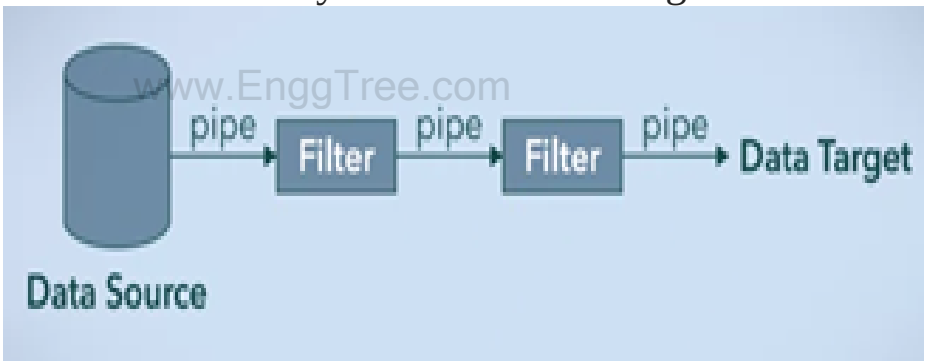
EnggTree.com

Many systems are required to transform streams of discrete data items, from input to output. Many types of transformations occur repeatedly in practice, and so it is desirable to create these as independent, reusable parts, Filters. (Len Bass, 2012)

Description of the Pattern

The pattern of interaction in the pipe-and-filter pattern is characterized by successive

transformations of streams of data. As you can see in the diagram, the data flows in one direction. It starts at a data source, arrives at a filter's input port(s) where processing is done at the component, and then, is passed via its output port(s) through a pipe to the next filter, and then eventually ends at the data target.



A single filter can consume data from, or produce data to, one or more ports. They can also run concurrently and are not dependent. The output of one filter is the input of another, hence, the order is very important.

A pipe has a single source for its input and a single target for its output. It preserves the sequence of data items, and it does not alter the data passing through.

Downloaded from EnggTree.com

EnggTree.com

Advantages of selecting the pipe and filter architecture are as follows:

· Ensures loose and flexible coupling of components, filters.

· Loose coupling allows filters to be changed without modifications to other filters.

· Conductive to parallel processing.

· Filters can be treated as black boxes. Users of the system don't need to know the logic behind the working of each filter.

· Re-usability. Each filter can be called and used over and over again.

www.EnggTree.com

However, there are a few drawbacks to this architecture and are discussed below:

· Addition of a large number of independent filters may reduce performance due to excessive computational overheads.

· Not a good choice for an interactive system.

· Pipe-and-fitter systems may not be appropriate for long-running computations.

Applications of the Pattern

Downloaded from EnggTree.com

EnggTree.com

In software engineering, a **pipeline** consists of a chain of processing elements (processes, threads, functions, etc.), arranged so that the output of each element is the input of the next. (Wiki, n.d.).

The architectural pattern is very popular and used in many systems, such as the text-based utilities in the UNIX operating system. Whenever different data sets need to be manipulated in different ways, you should consider using the pipe and filter architecture. More specific implementations are discussed below:

1. Compilers:

A compiler performs language transformation: Input is in language A and output is in language B. In order to do that the input goes through various stages inside the compiler — these stages form the pipeline. The most commonly used division consists of 3 stages: front-end, middle-end, and back-end.

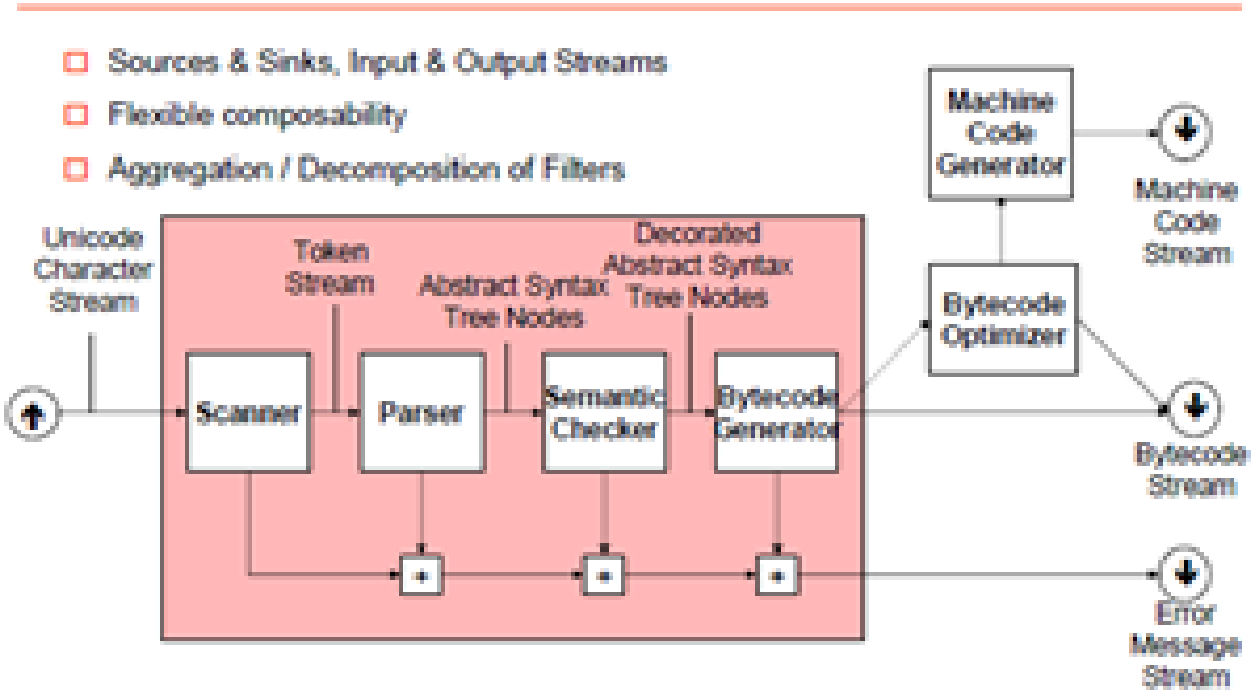
www.EnggTree.com

The front-end is responsible for parsing the input language and performing syntax and semantic and then transforms it into an intermediate language. The middle-end takes the intermediate representation and usually performs several optimization steps on it, the resulting transformed program in is passed to the back-end which transforms it into language B.

Each level consists of several steps as well, and everything together forms the pipeline of the compiler.

Downloaded from EnggTree.com

EnggTree.com



USER INTERFACE DESIGN – SOFTWARE ENGINEERING

www.EnggTree.com

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**

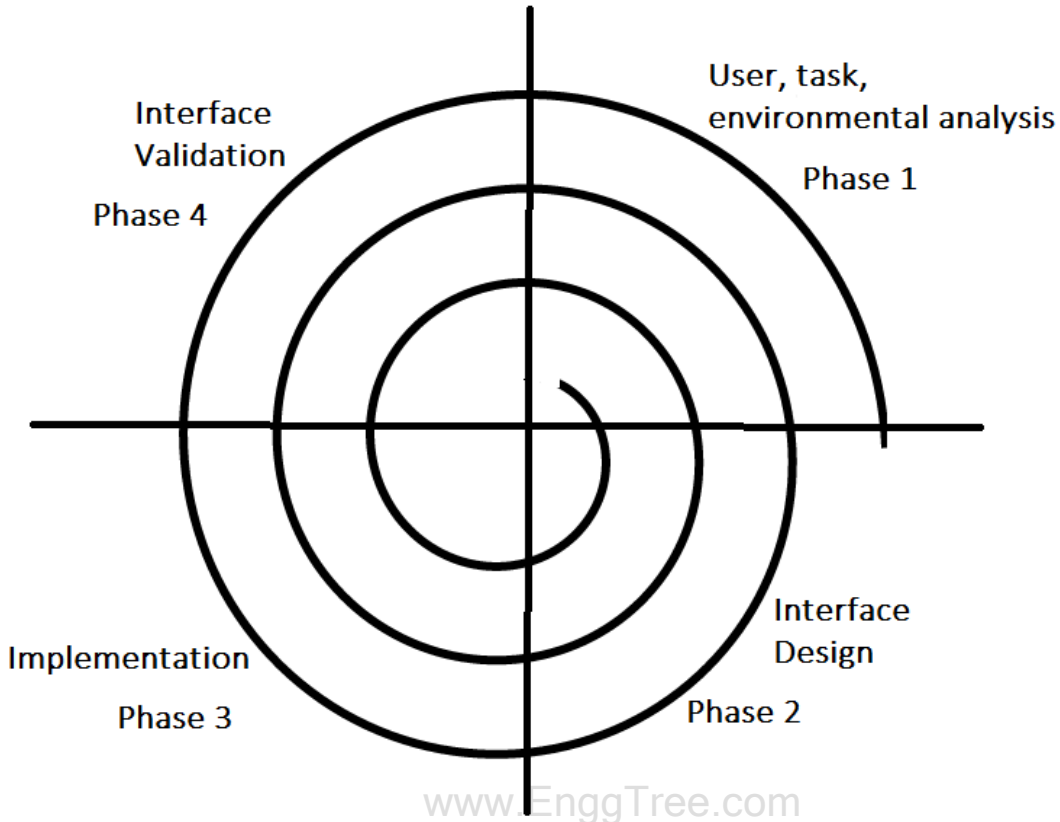
Types of User Interface

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.

Downloaded from EnggTree.com

EnggTree.com

User Interface Design Process



www.EnggTree.com

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?

Downloaded from EnggTree.com

EnggTree.com

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.

Interface Construction and Implementation

The implementation activity begins with the creation of a 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.

www.EnggTree.com

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.

User Interface Design Golden Rules

The following are the golden rules stated by Theo Mandel that must be followed during the design of the interface. **Place the user in control:**

1. **Define the interaction modes in such a way that does not force the user into unnecessary or undesired actions:** The user should be able to easily enter and exit the mode with little or no effort.
2. **Provide for flexible interaction:** Different people will use different interaction mechanisms, some might use keyboard commands, some might use mouse, some

Downloaded from EnggTree.com

EnggTree.com

might use touch screen, etc., Hence all interaction mechanisms should be provided.

3. **Allow user interaction to be interruptible and undoable:** When a user is doing a sequence of actions the user must be able to interrupt the sequence to do some other work without losing the work that had been done. The user should also be able to do undo operation.
4. **Streamline interaction as skill level advances and allow the interaction to be customized:** Advanced or highly skilled user should be provided a chance to customize the interface as user wants which allows different interaction mechanisms so that user doesn't feel bored while using the same interaction mechanism.
5. **Hide technical internals from casual users:** The user should not be aware of the internal technical details of the system. He should interact with the interface just to do his work.
6. **Design for direct interaction with objects that appear on-screen:** The user should be able to use the objects and manipulate the objects that are present on the screen to perform a necessary task. By this, the user feels easy to control over the screen.

www.EnggTree.com

Downloaded from EnggTree.com

EnggTree.com

UNIT-4

Testing-Unit testing-Black box testing-White box testing-Integration and System testing-Regression testing-Debugging-Program Analysis-Symbolic Execution-Model Checking-Case Study

1. TESTING:

Software Testing is a method to assess the functionality of the software program. The process checks whether the actual software matches the expected requirements and ensures the software is bug-free.

The purpose of software testing is to identify the errors, faults, or missing requirements in contrast to actual requirements. It mainly aims at measuring the specification, functionality, and performance of a software program or application.

www.EnggTree.com

1. UNIT TESTING

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

The purpose of unit testing is to validate that each unit of the software works as intendedand meets the requirements.

Unit testing is typically performed by developers, and it is performed early in the development process before the code is integrated and tested as a whole system.

Unit tests are automated and are run each time the code is changed to ensure that new code does not break existing functionality. Unit testsare

Downloaded from EnggTree.com

EnggTree.com

designed to validate the smallest possible unit of code, such as a function or a method, and test it in isolation from the rest of the system. This allows developers to quickly identify and fix any issues

www.EnggTree.com

Downloaded from EnggTree.com

EnggTree.com

Early in the development process, improving the overall quality of the software and reducing the time required for later testing.

Objective of Unit Testing:

The objective of Unit Testing is:

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

www.EnggTree.com

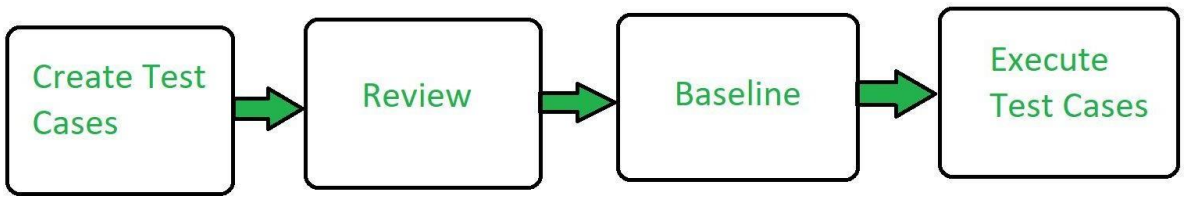
TYPES OF UNIT TESTING:

There are 2 types of Unit Testing:

1. **Manual**

2. Automated.

Workflow of Unit Testing:



Downloaded from EnggTree.com

EnggTree.com

Unit Testing Techniques:

There are 3 types of Unit Testing Techniques. They are

1. Black Box Testing: This testing technique is used in covering the unit tests for input, user interface, and outputparts.

2. White Box Testing: This technique is used in testing the functional behavior of the system by giving the input and checking the functionality output including the internal design structure and code of the modules.

3. Gray Box Testing: This technique is used in executing the relevant test cases, test methods, and test functions, and analyzing the code performance for the modules.

www.EnggTree.com

Downloaded from EnggTree.com

EnggTree.com

Advantages of Unit Testing:

1. Unit Testing allows developers to learn what functionality is provided by a unit and how to use it to gain a basic understanding of the unit API.
2. Unit testing allows the programmer to refine code and make sure the module works properly.
3. Unit testing enables testing parts of the project withoutwaiting for others to be completed.

Disadvantages of Unit Testing:

1. The process is time-consuming for writing the unit testcases.
2. Unit Testing will not cover all the errors in the module because there is a chance of having errors in the modules while doing integration testing.
3. Unit Testing is not efficient for checking the errors in the UI (User Interface) part of the module.
4. It requires more time for maintenance when the source codeis changed frequently.

www.EnggTree.com

BLACK BOX TESTING

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

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 atleast once.

Downloaded from EnggTree.com

EnggTree.com

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.

Types of Black Box Testing

There are many types of Black Box Testing but the following are the prominent ones –

- **Functional testing** – This black box testing type is related to the functional requirements of a system; it is done by software testers.
- **Non-functional testing** – This type of black box testing is not related to testing of specific functionality, but non-functional requirements such as performance, scalability, usability.
- **Regression testing** – Regression Testing is done after code fixes, upgrades or any other system maintenance to check the new code has not affected the existing code.

www.EnggTree.com

Tools used for Black Box Testing:

Tools used for Black box testing largely depend on the type of black box testing you are doing.

- For Functional/ Regression Tests you can use – QTP, Selenium
- For Non-Functional Tests, you can use – LoadRunner, Jmeter

WHITE BOX TESTING

White box testing techniques analyze the internal structures the used data structures, internal design, code structure, and the working of the software rather than just the functionality as in black box testing. It is

Downloaded from EnggTree.com

EnggTree.com

also called glass box testing or clear box testing or structural testing.

www.EnggTree.com

Downloaded from EnggTree.com

EnggTree.com

White Box Testing is also known as transparent testing or open box testing.

White box testing is a software testing technique that involves testing the internal structure and workings of a software application. The tester has access to the source code and uses this knowledge to design test cases that can verify the correctness of the software at thecode level.

White box testing is also known as structural testing or code-based testing, and it is used to test the software's internal logic, flow, and structure. The tester creates test cases to examine the code paths andlogic flows to ensure they meet the specified requirements.

Process of White Box Testing

1. **Input:** Requirements, Functional specifications, design documents, source code.
2. **Processing:** Performing risk analysis to guide through theentire process.
3. **Proper test planning:** Designing test cases to cover the entire code. Execute rinse-repeat until error-free software isreached. Also, the results are communicated.

www.EnggTree.com

4. **Output:** Preparing final report of the entire testing process.

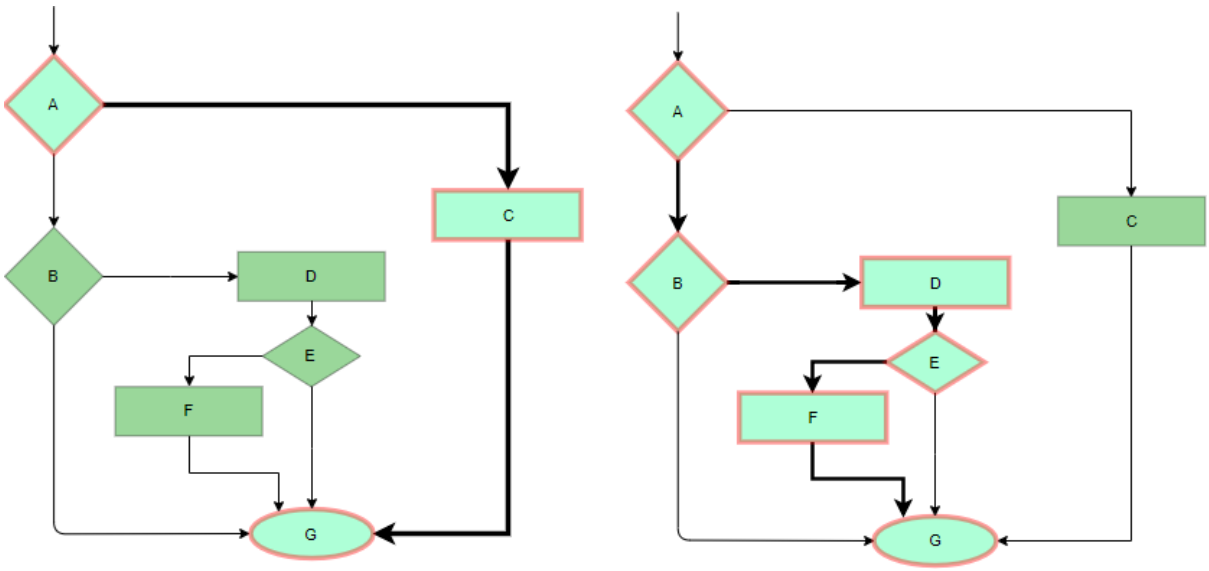
Testing Techniques

1. Statement Coverage

In this technique, the aim is to traverse all statements at least once. Hence, each line of code is tested. In the case of a flowchart, everynode must be traversed at least once. Since all lines of code are covered, it helps in pointing out faulty code.

Downloaded from EnggTree.com

EnggTree.com



Statement Coverage Example

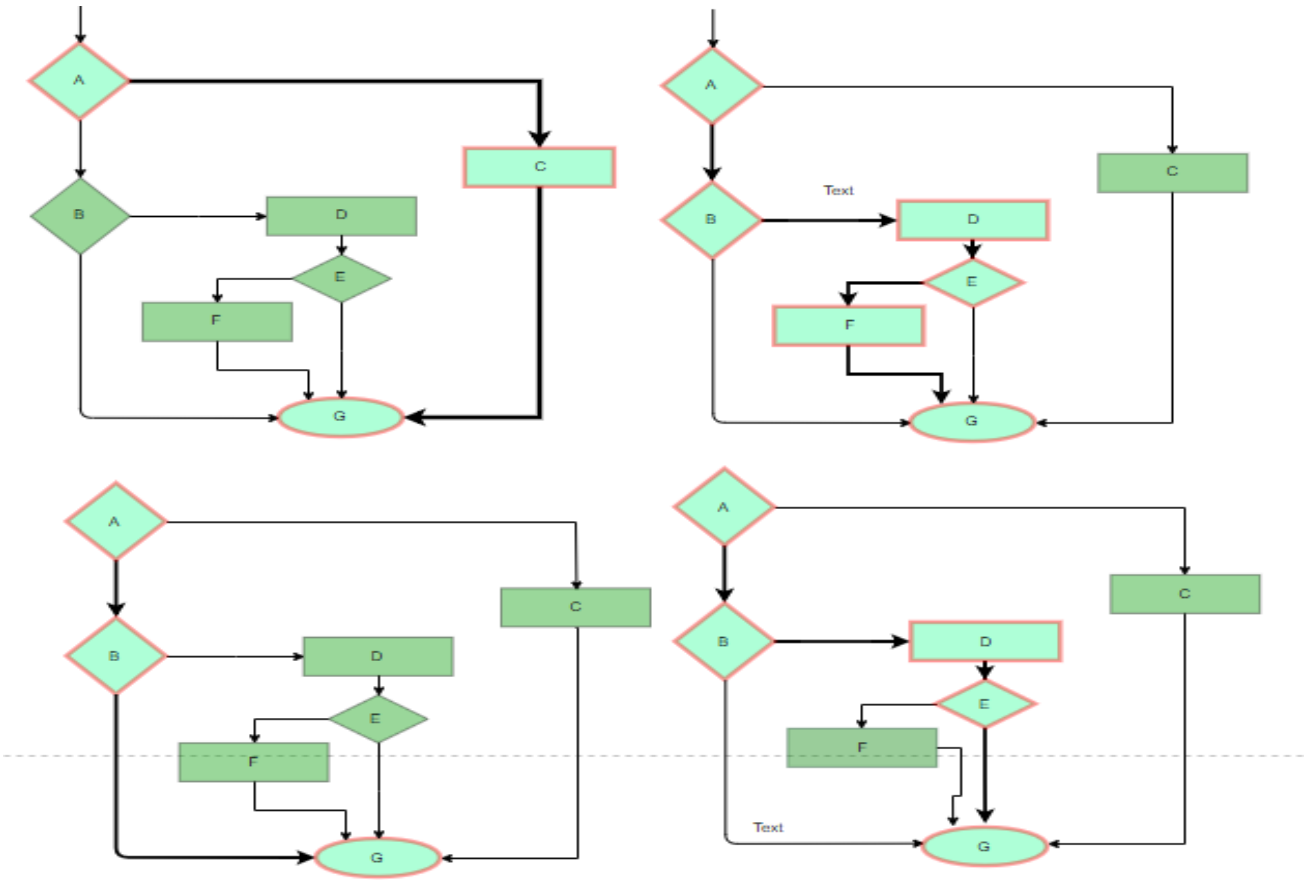
2. Branch Coverage:

In this technique, test cases are designed so that each branch from all decision points is traversed at least once. In a flowchart, all edges must be traversed at least once.

www.EnggTree.com

Downloaded from EnggTree.com

EnggTree.com



www.EnggTree.com

Downloaded from EnggTree.com

EnggTree.com

3.test cases are required such that all branches of all decisions are covered, i.e, all edges of the flowchart are covered

4. Condition Coverage

In this technique, all individual conditions must be covered as shown in the following example:

- READ X, Y
- IF(X == 0 || Y == 0)
- PRINT '0'
- #TC1 – X = 0, Y = 55
- #TC2 – X = 5, Y = 0

5. Multiple Condition Coverage

In this technique, all the possible combinations of the possible outcomes of conditions are tested at least once. Let's consider the following example:

- READ X, Y
- IF(X == 0 || Y == 0)
- PRINT '0'
- #TC1: X = 0, Y = 0
- #TC2: X = 0, Y = 5
- #TC3: X = 55, Y = 0
- #TC4: X = 55, Y = 5

www.EnggTree.com

6.Basis Path Testing

In this technique, control flow graphs are made from code or flowchart and then Cyclomatic complexity is calculated which defines the number of independent paths so that the minimal number of test cases can be designed for each independent path. **Steps:**

- Make the corresponding control flow graph
- Calculate the cyclomatic complexity
- Find the independent paths
- Design test cases corresponding to each independent path
- V(G) = P + 1, where P is the number of predicate nodes in the flow graph
- V(G) = E – N + 2, where E is the number of edges and N is the total number of nodes

Downloaded from EnggTree.com

EnggTree.com

- V(G) = Number of non-overlapping regions in the graph
- #P1: 1 – 2 – 4 – 7 – 8
- #P2: 1 – 2 – 3 – 5 – 7 – 8
- #P3: 1 – 2 – 3 – 6 – 7 – 8
- #P4: 1 – 2 – 4 – 7 – 1 – . . . – 7 – 8

3. Loop Testing

Loops are widely used and these are fundamental to many algorithms hence, their testing is very important. Errors often occur at the beginnings and ends of loops.

- **Simple loops:** For simple loops of size n, test cases are designed that:

1. Skip the loop entirely
2. Only one pass through the loop
3. 2 passes

www.EnggTree.com

Downloaded from EnggTree.com

EnggTree.com

4. m passes, where m < n
5. n-1 ans n+1 passes

- **Nested loops:** For nested loops, all the loops are set to their minimum count, and we start from the innermost loop. Simple loop tests are conducted for the innermost loop and this is worked outwards till all the loops have been tested.
- **Concatenated loops:** Independent loops, one after another. Simple loop tests are applied for each. If they're not independent, treat them like nesting.

White Testing is performed in 2 Steps

1. Tester should understand the code well
2. Tester should write some code for test cases and executethem

Tools required for White box testing:

- PyUnit
- Sqlmap
- Nmap
- Parasoft Jtest
- Nunit
- VeraUnit
- CppUnit
- Bugzilla
- Fiddler
- JSUnit.net
- OpenGrok
- Wireshark
- HP Fortify
- CSUnit

www.EnggTree.com

Advantages of Whitebox Testing

1. **Thorough Testing**: White box testing is thorough as the entire code and structures are tested.
2. **Code Optimization:** It results in the optimization of code

Downloaded from EnggTree.com

EnggTree.com

removing errors and helps in removing extra lines of code.

www.EnggTree.com

Downloaded from EnggTree.com

EnggTree.com

3. **Early Detection of Defects:** It can start at an earlier stage as it doesn't require any interface as in the case of black box testing.
4. **Integration with SDLC:** White box testing can be easily started in Software Development Life Cycle.
5. **Detection of Complex Defects:** Testers can identify defects that cannot be detected through other testing techniques.
6. **Comprehensive Test Cases:** Testers can create more comprehensive and effective test cases that cover all codepaths.
7. Testers can ensure that the code meets coding standards andis optimized for performance.

Disadvantages of White box Testing

1. **Programming Knowledge and Source Code Access:** Testers need to have programming knowledge and access to the source code to perform tests.
2. **Overemphasis on Internal Workings:** Testers may focus too much on the internal workings of the software and maymiss external issues.
3. **Bias in Testing:** Testers may have a biased view of the software since they are familiar with its internal workings.
4. **Test Case Overhead:** Redesigning code and rewriting code needs test cases to be written again.
5. **Dependency on Tester Expertise:** Testers are required to have in-depth knowledge of the code and programming language as opposed to black-box testing.
6. **Inability to Detect Missing Functionalities:** Missing functionalities cannot be detected as the code that exists istested.
7. **Increased Production Errors:** High chances of errors in production.

Downloaded from EnggTree.com

EnggTree.com

Differences between Black Box Testing vs White BoxTesting:

www.EnggTree.com

Downloaded from EnggTree.com

EnggTree.com

Black Box Testing	White Box Testing
It is a way of software testing in which the internal structure or the program or the code is hidden and nothing is known about it.	It is a way of testing the software in which the tester hasknowledge about the internal structure or the code or the program of the software.
Implementation of code is not needed for black box testing.	Code implementation is necessary for white box testing.
It is mostly done by software testers.	It is mostly done by software developers.
No knowledge of implementation is needed.	Knowledge of implementationis required.
It can be referred to as outer or external software testing.	It is the inner or the internal software testing.
It is a functional test of the software.	It is a structural test of the software.
This testing can be initiatedbased on the requirement specifications document.	This type of testing of softwareis started after a detail design document.
No knowledge of programmingis required.	It is mandatory to have knowledge of programming.
It is the behavior testing of the software.	It is the logic testing of the software.

www.EnggTree.com

Downloaded from EnggTree.com

EnggTree.com

Black Box Testing	White Box Testing
It is applicable to the higher levels of testing of software.	It is generally applicable to the lower levels of software testing.
It is also called closed testing.	It is also called as clear box testing.
It is least time consuming.	It is most time consuming.
It is not suitable or preferred for algorithm testing.	It is suitable for algorithm testing.
Can be done by trial and error ways and methods.	Data domains along with inner or internal boundaries can be better tested.
Example: Search something on google by using keywords	Example: By input to check and verify loops
Black-box test design techniques- • Decision table testing • All-pairs testing • Equivalence partitioning • Error guessing	White-box test design techniques- • Control flow testing • Data flow testing • Branch testing
Types of Black Box Testing: • Functional Testing • Non-functional testing • Regression Testing	Types of White Box Testing: • Path Testing • Loop Testing • Condition testing

www.EnggTree.com

Downloaded from EnggTree.com

EnggTree.com

Black Box Testing	White Box Testing
It is less exhaustive as compared to white box testing.	It is comparatively more exhaustive than black box testing.

www.EnggTree.com

Downloaded from EnggTree.com

EnggTree.com

2. INTEGRATION TESTING AND SYSTEM TESTING

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.

www.EnggTree.com

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 test approaches – There are four types of integration testing approaches. Those approaches are the following:

1. Big-Bang Integration Testing – It is the simplest integration testing approach, where all the modules are combined and the functionality is verified after the completion of individual module

Downloaded from EnggTree.com

EnggTree.com

Testing. In simple words, all the modules of the system are simply put together and tested.

This approach is practicable only for very small systems. If an error is found during the integration testing, it is very difficult to localize the error as the error may potentially belong to any of the modules being integrated. So, debugging errors reported during Big Bang integration testing is very expensive to fix.

Big-bang integration testing is a software testing approach in which all components or modules of a software application are combined andtested at once.

This approach is typically used when the software components have a low degree of interdependence or when there are constraints in the development environment that prevent testing individual components.

The goal of big-bang integration testing is to verify the overall functionality of the system and to identify anyintegration problems that arise when the components are combined.

While big-bang integration testing can be useful in some situations, it can also be a high-risk approach, as the complexity of the system and the number of interactions between components can make it difficult to identify and diagnose problems.

Advantages:

1. It is convenient for small systems.
2. Simple and straightforward approach.
3. Can be completed quickly.
4. Does not require a lot of planning or coordination.
5. May be suitable for small systems or projects with a low degree of interdependence between components.

Disadvantages:

1. There will be quite a lot of delay because you would have to wait for all the modules to be integrated.
2. High-risk critical modules are not isolated and tested on priority since all modules are tested at once.
3. Not Good for long projects.
4. High risk of integration problems that are difficult to dentify and diagnose.
5. This can result in long and complex debugging

Downloaded from EnggTree.com

EnggTree.com

andtroubleshooting efforts.

6. This can lead to system downtime and increaseddevelopment costs.

www.EnggTree.com

Downloaded from EnggTree.com

EnggTree.com

7. May not provide enough visibility into the interactions and data exchange between components.
8. This can result in a lack of confidence in the system's stability and reliability.
9. This can lead to decreased efficiency and productivity.
10. This may result in a lack of confidence in the development team.
11. This can lead to system failure and decreased user satisfaction.

2. Bottom-Up Integration Testing – In bottom-up testing, each module at lower levels are tested with higher modules until all modules are tested.

3. The primary purpose of this integration testing is that each subsystem tests the interfaces among various modules making up the subsystem. This integration testing uses test drivers todrive and pass appropriate data to the lower-level modules.

Advantages:

- In bottom-up testing, no stubs are required.
- A principal advantage of this integration testing is that several disjoint subsystems can be tested simultaneously.
- It is easy to create the test conditions.
- Best for applications that uses bottom up design approach.
- It is Easy to observe the test results.

Disadvantages:

- Driver modules must be produced.
- In this testing, the complexity that occurs when the system is made up of a large number of small subsystems.
- As Far modules have been created, there is no working model can be represented.

4. Top-Down Integration Testing – Top-down integration testing technique is used in order to simulate the behaviour of the lower-level modules that are not yet integrated. In this integration testing, testing takes place from top to bottom. First, high-level modules are tested and then low-level modules and finally integrating the low-level modules to a high level to ensure the system is working as intended.

Advantages:

- Separately debugged module.

Downloaded from EnggTree.com

EnggTree.com

- Few or no drivers needed.

www.EnggTree.com

Downloaded from EnggTree.com

EnggTree.com

- It is more stable and accurate at the aggregate level.
- Easier isolation of interface errors.
- In this, design defects can be found in the early stages.

Disadvantages:

- Needs many Stubs.
- Modules at lower level are tested inadequately.
- It is difficult to observe the test output.
- It is difficult to stub design.

5. Mixed Integration Testing – A mixed integration testing is also called sandwiched integration testing. A mixed integration testing follows a combination of top down and bottom-up testing approaches. In top-down approach, testing can start only after the top-level module have been coded and unit tested.

6. In bottom-up approach, testing can start only after the bottom level modules are ready. This sandwich or mixed approach overcomes this shortcoming of the top-down and bottom-up approaches. It is also called the hybrid integration testing.also, stubs and drivers are used in mixed integration testing. **Advantages:**

- Mixed approach is useful for very large projects having several sub projects.
- This Sandwich approach overcomes this shortcoming of the top-down and bottom-up approaches.
- Parallel test can be performed in top and bottom layer tests.

Disadvantages:

- For mixed integration testing, it requires very high cost because one part has a Top-down approach while another part has a bottom-up approach.
- This integration testing cannot be used for smaller systems with huge interdependence between different modules.

Applications:

1. **Identify the components:** Identify the individual components of your application that need to be integrated. This could include the frontend, backend, database, and anythird-party services.
2. **Create a test plan:** Develop a test plan that outlines the scenarios and test cases that need to be executed to validate the integration points between the different components.

Downloaded from EnggTree.com

EnggTree.com

This could include testing data flow, communication protocols, and error handling.

3. **Set up test environment:** Set up a test environment that mirrors the production environment as closely as possible. This will help ensure that the results of your integration tests are accurate and reliable.
4. **Execute the tests:** Execute the tests outlined in your test plan, starting with the most critical and complex scenarios.Be sure to log any defects or issues that you encounter during testing.
5. **Analyze the results:** Analyze the results of your integration tests to identify any defects or issues that need to be addressed. This may involve working with developers to fixbugs or make changes to the application architecture.
6. **Repeat testing:** Once defects have been fixed, repeat the integration testing process to ensure that the changes havebeen successful and that the application still works as expected.

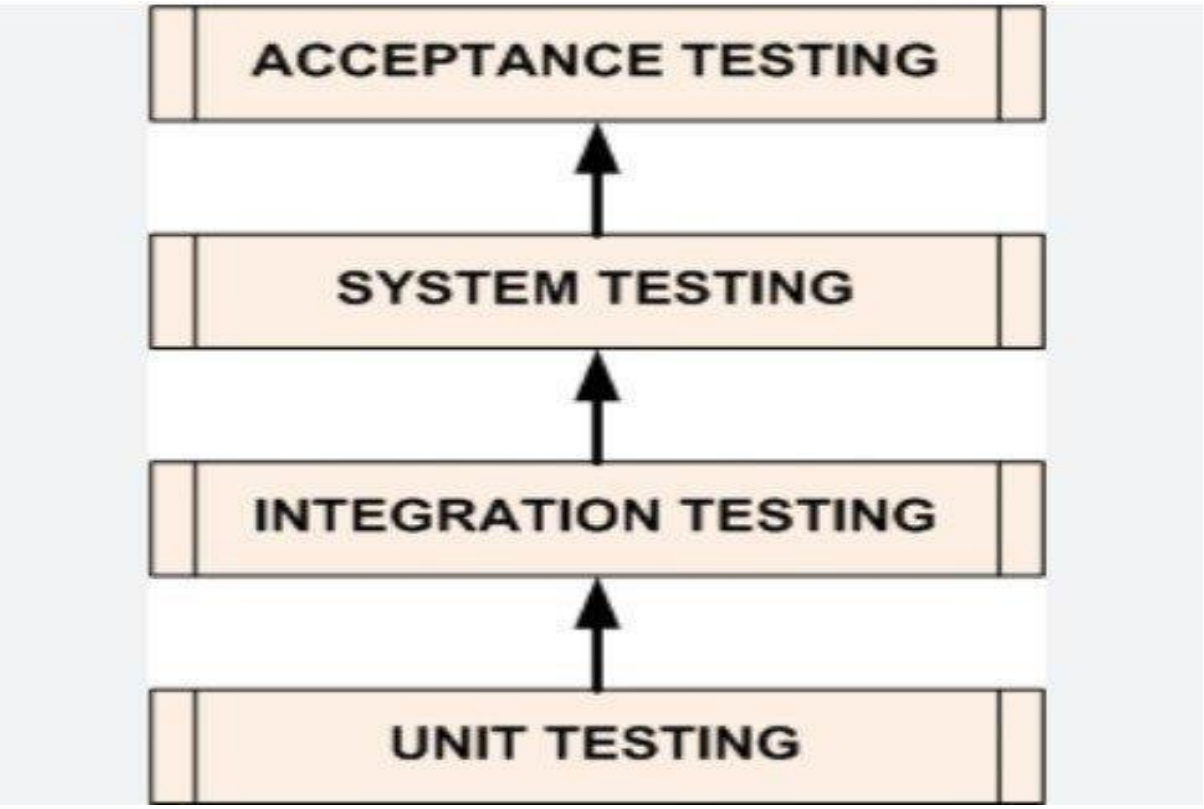
System testing

www.EnggTree.com

A type of Software testing, System testing comes at the third level after Unit testing and Integration testing. The goal of the system testing is to compare the functional and non-functional features of the system against the user requirements.

Downloaded from EnggTree.com

EnggTree.com



Stages of Software Testing

www.EnggTree.com

Once all the sub-systems or modules integrate to build one application, testers perform System testing to check for potential functional and non-functional irregularities. Overall, System testing checks for the system's design and behavior as per customer
Expectations. It is a black box testing technique.

Often, your QA team will rely on System Requirement Specifications (SRS), Functional Requirement Specifications (FRS),
or a mix of both. We will discuss these two options later in this blog.

Types of System Testing

System testing covers the end-to-end analysis of a software
application in terms of its functionality and user experience. Hence, you can categorize it into two parts—functional and non-functional.

Functional Testing

Functional testing validates the functional aspects of a software
system against the user requirements and functional specifications. Itchecks for user experience or interface, API integration, database,

Downloaded from EnggTree.com

EnggTree.com

security and privacy, and server communication, as mentioned in the user document.

One example of functional testing is checking for the login functionality using the right credentials. The system should not allow you to sign in if your username and password are wrong or not present in the database.

Non-Functional Testing

Non-functional testing checks for non-functional aspects of software, such as performance, reliability, usability, and application readiness. It intends to assess a system's performance per the non-functional conditions that never appear in the functional tests. Often, non- functional testing is important to check for security, application load capability, and utility to measure user satisfaction.

One example of non-functional testing is checking for the number of users the system can handle at a time. It helps to determine the performance and usability of the system under high traffic.

Advantages of System Testing

www.EnggTree.com

- End-to-end test that involves all the software components as a whole and checks for defects
- Examine the software from a user point of view and simulate real-life scenarios
- Covers both functional and non-functional testing elements such as performance, usability, regression testing, and more
- No need for internal code knowledge

Downloaded from EnggTree.com

EnggTree.com

Differences between System Testing and Integration Testing

	System Testing	Integration Testing
Purpose	Testing the entire software as a whole and understanding if it meets the user requirements	Testing several modules together and understanding how they interface with each other
Executed by	Test Engineers	QA Testers, Developers and Test Engineers
Type of testing	Both functional and non-functional tests such as performance, security, regression testing, unit testing, etc	Only functional tests. Can be performed in different approaches like top-down, bottom-up, big-bang, and more
Technique	Black-box testing	Grey-box testing
When to perform	After Integration Testing	Before System Testing and after Unit Testing

www.EnggTree.com

Downloaded from EnggTree.com

EnggTree.com

Advantages	Involves all the software components andchecks for bugs Examine the software from a user point of viewand simulate real-life scenarios Covers both functional and non-functional testing elements No need for internalcode knowledge	• Helps to figure out how different modules communicate with each other • Finds defects in the interface between various components Can be performed bya wider variety of staff members. There are different approaches with different benefits toconsider when performing integration testing

www.EnggTree.com

Downloaded from EnggTree.com

EnggTree.com

www.EnggTree.com

Downloaded from EnggTree.com

EnggTree.com

3. REGRESSION TESTING, DEBUGGING, PROGRAM ANALYSIS

It is the process of testing the modified parts of the code and the parts that might get affected due to the modifications to ensure thatno new errors have been introduced in the software after the modifications have been made. Regression means the return of something and in the software field, it refers to the return of a bug.

When to do regression testing?

- When a new functionality is added to the system and the code has been modified to absorb and integrate that functionality with the existing code.
- When some defect has been identified in the software and the code is debugged to fix it.
- When the code is modified to optimize its working.

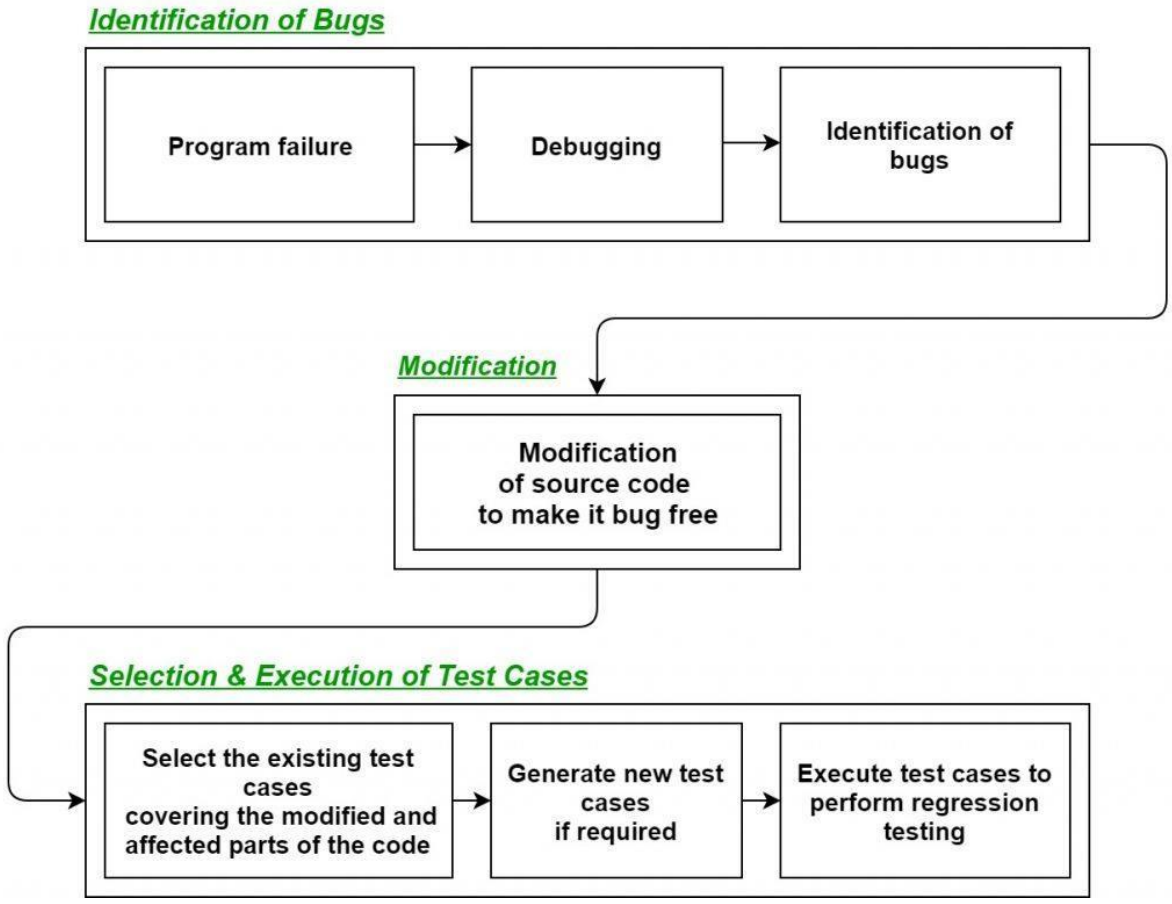
Process of Regression testing:

0 seconds of 17 seconds Volume 0% Firstly, whenever we make some changes to the source code for any reason like adding new functionality, optimization, etc. then our program when executed fails in the previously designed test suite for obvious reasons. After the failure, the source code is debugged in order to identify the bugs in the program. After identification of the bugs in the source code, appropriate modifications are made. Then appropriate test cases are selected from the already existing test suite which covers all the modified and affected parts of the source code. We can add new test cases if required. In the end, regression testing is performed using the selected test cases.

www.EnggTree.com

Downloaded from EnggTree.com

EnggTree.com



www.EnggTree.com

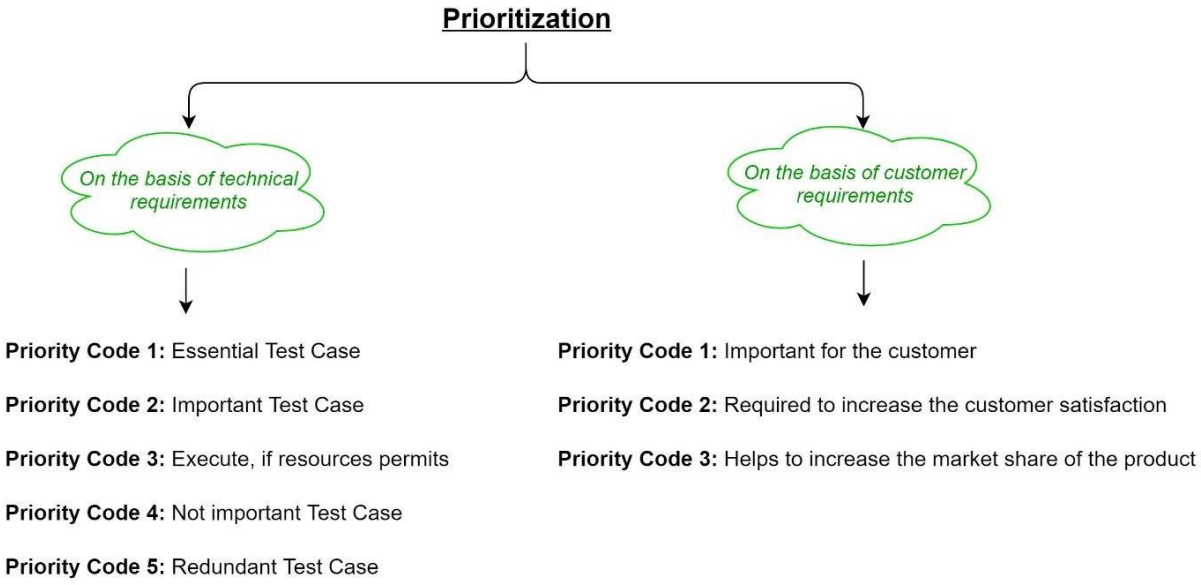
Techniques for the selection of Test cases for Regression Testing:

- **Select all test cases:** In this technique, all the test cases are selected from the already existing test suite. It is the simplest and safest technique but not much efficient.
- **Select test cases randomly:** In this technique, test cases are selected randomly from the existing test-suite, but it is only useful if all the test cases are equally good in their fault detection capability which is very rare. Hence, it is not used in most of the cases.
- **Select modification traversing test cases:** In this technique, only those test cases are selected which covers and tests the modified portions of the source code the parts which are affected by these modifications.
- **Select higher priority test cases:** In this technique, priority codes are assigned to each test case of the test suite based upon their bug detection capability, customer requirements, etc. After assigning the priority codes, test cases with the highest priorities are selected for the process of regression testing. The test case with the highest priority has the highest

Downloaded from EnggTree.com

EnggTree.com

rank. For example, test case with priority code 2 is less important than test case with priority code 1.



Tools for regression testing:

In regression testing, we generally select the test cases from the existing test suite itself and hence, we need not compute their expected output, and it can be easily automated due to this reason. Automating the process of regression testing will be very much effective and time saving. Most commonly used tools for regression testing are:

- Selenium
- WATIR (Web Application Testing In Ruby)
- QTP (Quick Test Professional)
- RFT (Rational Functional Tester)
- Winrunner
- Silktest

Advantages of Regression Testing:

- It ensures that no new bugs has been introduced after adding new functionalities to the system.
- As most of the test cases used in Regression Testing are selected from the existing test suite, and we already know their expected outputs. Hence, it can be easily automated by the automated tools.
- It helps to maintain the quality of the source code.

Downloaded from EnggTree.com

EnggTree.com

Disadvantages of Regression Testing:

- It can be time and resource consuming if automated tools are not used.
- It is required even after very small changes in the code.

DEBUGGING

Debugging is the process of identifying and resolving errors, or bugs, in a software system. It is an important aspect of software engineering because bugs can cause a software system to malfunction, and can lead to poor performance or incorrect results. Debugging can be a time-consuming and complex task, but it is essential for ensuring that a software system is functioning correctly.

There are several common methods and techniques used in debugging, including:

1. **Code Inspection:** This involves manually reviewing the source code of a software system to identify potential bugs or errors.
2. **Debugging Tools:** There are various tools available for debugging such as debuggers, trace tools, and profilers that can be used to identify and resolve bugs.
3. **Unit Testing:** This involves testing individual units or components of a software system to identify bugs or errors.
4. **Integration Testing:** This involves testing the interactions between different components of a software system to identify bugs or errors.
5. **System Testing:** This involves testing the entire software system to identify bugs or errors.
6. **Monitoring**: This involves monitoring a software system for unusual behavior or performance issues that can indicate the presence of bugs or errors.
7. **Logging**: This involves recording events and messages related to the software system, which can be used to identify bugs or errors.

It is important to note that debugging is an iterative process, and it may take multiple attempts to identify and resolve all bugs in a software system. Additionally, it is important to have a well-defined

Downloaded from EnggTree.com

EnggTree.com

process in place for reporting and tracking bugs, so that they can be effectively managed and resolved.

In summary, debugging is an important aspect of software engineering, it's the process of identifying and resolving errors, or bugs, in a software system.

There are several common methods and techniques used in debugging, including code inspection, debugging tools, unit testing, integration testing, system testing, monitoring, and logging. It is an iterative process that may take multiple attempts to identify and resolve all bugs in a software system.

In the context of software engineering, debugging is the process of fixing a bug in the software. In other words, it refers to identifying, analyzing, and removing errors.

This activity begins after the software fails to execute properly and concludes by solving the problem and successfully testing the software.

It is considered to be an extremely complex and tedious task because errors need to be resolved at all stages of debugging.

A better approach is to run the program within a debugger, which is a specialized environment for controlling and monitoring the execution of a program.

The basic functionality provided by a debugger is the insertion of breakpoints within the code. When the program is executed within the debugger, it stops at each breakpoint. Many IDEs,such as Visual C++ and C-Builder provide built-in debuggers.

Debugging Process: The steps involved in debugging are:

- Problem identification and report preparation.
- Assigning the report to the software engineer defect to verify that it is genuine.
- Defect Analysis using modeling, documentation, finding and testing candidate flaws, etc.
- Defect Resolution by making required changes to the system.
- Validation of corrections.

The debugging process will always have one of two outcomes:

1. The cause will be found and corrected.
2. The cause will not be found.

Downloaded from EnggTree.com

EnggTree.com

Later, the person performing debugging may suspect a cause, design a test case to help validate that suspicion, and work toward error correction in an iterative fashion.

www.EnggTree.com

Downloaded from EnggTree.com

EnggTree.com

During debugging, we encounter errors that range from mildly annoying to catastrophic.

As the consequences of an error increase, the amount of pressure to find the cause also increases. Often, pressure sometimes forces a software developer to fix one error and at the same time introduce two more.

Debugging Approaches/Strategies:

1. **Brute Force:** Study the system for a longer duration to understand the system. It helps the debugger to construct different representations of systems to be debugged depending on the need. A study of the system is also done actively to find recent changes made to the software.
2. **Backtracking:** Backward analysis of the problem which involves tracing the program backward from the location of the failure message to identify the region of faulty code. A detailed study of the region is conducted to find the cause of defects.
3. **Forward analysis** of the program involves tracing the program forwards using breakpoints or print statements at different points in the program and studying the results. The region where the wrong outputs are obtained is the region that needs to be focused on to find the defect.
4. **Using** A debugging**experience** with the software debug the software with similar problems in nature. The success of this approach depends on the expertise of the debugger.
5. **Cause elimination:** it introduces the concept of binary partitioning. Data related to the error occurrence are organized to isolate potential causes.
6. **Static analysis:** Analyzing the code without executing it to identify potential bugs or errors. This approach involves analyzing code syntax, data flow, and control flow.
7. **Dynamic analysis:** Executing the code and analyzing its behavior at runtime to identify errors or bugs. This approach involves techniques like runtime debugging and profiling.
8. **Collaborative debugging:** Involves multiple developers working together to debug a system. This approach is helpful in situations where multiple modules or components are

Downloaded from EnggTree.com

EnggTree.com

involved, and the root cause of the error is not clear.

www.EnggTree.com

Downloaded from EnggTree.com

EnggTree.com

9. **Logging and Tracing:** Using logging and tracing tools to identify the sequence of events leading up to the error. This approach involves collecting and analyzing logs and traces generated by the system during its execution.
10. **Automated Debugging:** The use of automated tools and techniques to assist in the debugging process. These tools can include static and dynamic analysis tools, as well as tools that use machine learning and artificial intelligence to identify errors and suggest fixes.

Debugging Tools:

A debugging tool is a computer program that is used to test and debug other programs. A lot of public domain software like gdb and dbx are available for debugging. They offer console-based command-line interfaces. Examples of automated debugging tools include code-based tracers, profilers, interpreters, etc. Some of the widely used debuggers are:

- Radare2
- WinDbg
- Valgrind

www.EnggTree.com

Difference Between Debugging and Testing:

Debugging is different from testing. Testing focuses on finding bugs, errors, etc whereas debugging starts after a bug has been identified in the software. Testing is used to ensure that the program is correct and it was supposed to do with a certain minimum success rate. Testing can be manual or automated. There are several different types of testing unit testing, integration testing, alpha, and beta testing, etc. Debugging requires a lot of knowledge, skills, and expertise. It can be supported by some automated tools available but is more of a manual process as every bug is different and requires a different technique, unlike a pre-defined testing mechanism.

Advantages of Debugging:

Several advantages of debugging in software engineering:

1. **Improved system quality:** By identifying and resolving bugs, a software system can be made more reliable and efficient, resulting in improved overall quality.
2. **Reduced system downtime:** By identifying and resolving bugs, a software system can be made more stable and less

Downloaded from EnggTree.com

EnggTree.com

likely to experience downtime, which can result in improved availability for users.

3. **Increased user satisfaction**: By identifying and resolving bugs, a software system can be made more user-friendly and better able to meet the needs of users, which can result in increased satisfaction.
4. **Reduced development costs:** Identifying and resolving bugs early in the development process, can save time and resources that would otherwise be spent on fixing bugs later in the development process or after the system has been deployed.
5. **Increased security**: By identifying and resolving bugs that could be exploited by attackers, a software system can be made more secure, reducing the risk of security breaches.
6. **Facilitates change:** With debugging, it becomes easy to make changes to the software as it becomes easy to identify and fix bugs that would have been caused by the changes.
7. **Better understanding of the system:** Debugging can help developers gain a better understanding of how a software system works, and how different components of the system interact with one another.
8. **Facilitates testing:** By identifying and resolving bugs, it makes it easier to test the software and ensure that it meets the requirements and specifications.

In summary, debugging is an important aspect of software engineering as it helps to improve system quality, reduce system downtime, increase user satisfaction, reduce development costs, increase security, facilitate change, a better understanding of the system, and facilitate testing.

Disadvantages of Debugging:

While debugging is an important aspect of software engineering, there are also some disadvantages to consider:

1. **Time-consuming:** Debugging can be a time-consuming process, especially if the bug is difficult to find or reproduce. This can cause delays in the development process and add to the overall cost of the project.

Downloaded from EnggTree.com

EnggTree.com

2. **Requires specialized skills:** Debugging can be a complex task that requires specialized skills and knowledge. This can be a challenge for developers who are not familiar with the tools and techniques used in debugging.
3. **Can be difficult to reproduce:** Some bugs may be difficult to reproduce, which can make it challenging to identify and resolve them.
4. **Can be difficult to diagnose:** Some bugs may be caused by interactions between different components of a software system, which can make it challenging to identify the root cause of the problem.
5. **Can be difficult to fix:** Some bugs may be caused by fundamental design flaws or architecture issues, which can be difficult or impossible to fix without significant changes to the software system.
6. **Limited insight**: In some cases, debugging tools can only provide limited insight into the problem and may not provide enough information to identify the root cause of the problem.

www.EnggTree.com

7. **Can be expensive:** Debugging can be an expensive process, especially if it requires additional resources such as specialized debugging tools or additional development time.

In summary, debugging is an important aspect of software engineering but it also has some disadvantages, it can be time-consuming, requires specialized skills, can be difficult to reproduce, diagnose, and fix, may have limited insight, and can be expensive.

PROGRAM ANALYSIS

Program Analysis Tool is an automated tool whose input is the source code or the executable code of a program and the output is the observation of characteristics of the program.

It gives various characteristics of the program such as its size, complexity, adequacy of commenting, adherence to programming standards and many other characteristics. These tools are essential to software engineering because they help programmers comprehend,

Downloaded from EnggTree.com

EnggTree.com

Improve and maintain software systems over the course of the whole development life cycle.

Importance of Program Analysis Tools

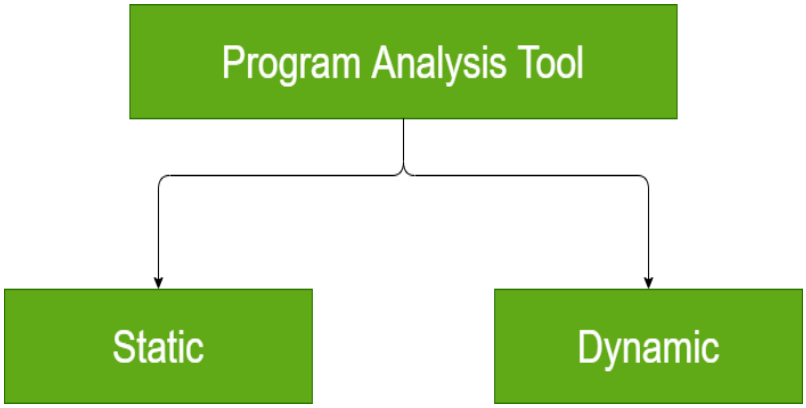
1. **Finding faults and Security Vulnerabilities in the Code:** Automatic programme analysis tools can find and highlight possible faults, security flaws and bugs in the code. This lowers the possibility that bugs will get it into production by assisting developers in identifying problems early in the process.
2. **Memory Leak Detection:** Certain tools are designed specifically to find memory leaks and inefficiencies. By doing so, developers may make sure that their software doesn't gradually use up too much memory.
3. **Vulnerability Detection:** Potential vulnerabilities like buffer overflows, injection attacks or other security flaws can be found using programme analysis tools, particularly those that are security-focused. For the development of reliable and secure software, this is essential.
4. **Dependency analysis:** By examining the dependencies among various system components, tools can assist developers in comprehending and controlling the connections between modules. This is necessary in order to make well-informed decisions during refactoring.
5. **Automated Testing Support:** To automate testing procedures, CI/CD pipelines frequently combine programme analysis tools. Only well-tested, high-quality code is released into production thanks to this integration, helping in identifying problems early in the development cycle.

Classification of Program Analysis Tools

Program Analysis Tools are classified into two categories:

Downloaded from EnggTree.com

EnggTree.com



1. Static Program Analysis Tools

Static Program Analysis Tool is such a program analysis tool that evaluates and computes various characteristics of a software product without executing it. Normally, static program analysis tools analyze some structural representation of a program to reach a certain analytical conclusion.

Basically some structural properties are analyzed using static program analysis tools. The structural properties that are usually analyzed are:

1. Whether the coding standards have been fulfilled or not.
2. Some programming errors such as uninitialized variables.
3. Mismatch between actual and formal parameters.
4. Variables that are declared but never used.

Code walkthroughs and code inspections are considered as static analysis methods but static program analysis tool is used to designate automated analysis tools. Hence, a compiler can be considered as a static program analysis tool.

2. Dynamic Program Analysis Tools

Dynamic Program Analysis Tool is such type of program analysis tool that require the program to be executed and its actual behavior to be observed. A dynamic program analyzer basically implements the code. It adds additional statements in the source code to collect the traces of program execution.

When the code is executed, it allows us to observe the behavior of the software for different test cases. Once the software is tested and its behavior is observed, the dynamic program analysis tool performs a post execution analysis and produces reports which describe the structural coverage that has been achieved by the complete testing process for the program.

Downloaded from EnggTree.com

EnggTree.com

For example, the post execution dynamic analysis report may provide data on extent statement, branch and path coverage

www.EnggTree.com

Downloaded from EnggTree.com

EnggTree.com

achieved. The results of dynamic program analysis tools are in the form of a histogram or a pie chart. It describes the structural coverage obtained for different modules of the program.

The output of a dynamic program analysis tool can be stored and printed easily and provides evidence that complete testing has been done. The result of dynamic analysis is the extent of testing performed as whitebox testing. If the testing result is not satisfactory then more test cases are designed and added to the test scenario. Also dynamic analysis helps in elimination of redundant test cases.

www.EnggTree.com

Downloaded from EnggTree.com

EnggTree.com

4.SYMBOLIC EXECUTION

Symbolic execution is a software testing technique that is useful to aid

the generation of test data and in proving the program quality.

Steps to use Symbolic Execution:

- The execution requires a selection of paths that are exercised bya set of data values. A program, which is executed using actual data, results in the output of a series of values.
- In symbolic execution, the data is replaced by symbolic values with set of expressions, one expression per output variable.
- The common approach for symbolic execution is to perform an analysis of the program, resulting in the creation of a flow graph.
- The flowgraph identifies the decision points and the assignments associated with each flow. By traversing the flow graph from an entry point, a list of assignment statements and branch predicates is produced.

www.EnggTree.com

Issues with Symbolic Execution:

- Symbolic execution cannot proceed if the number of iterations in the loop is known.
- The second issue is the invocation of any out-of-line code or module calls.
- Symbolic execution cannot be used with arrays.
- The symbolic execution cannot identify of infeasible paths.

Symbolic Execution Application:

- Path domain checking

Downloaded from EnggTree.com

EnggTree.com

- Test Data generation
- Partition analysis
- Symbolic debugging

MODEL CHECKING

Model checking is the most successful approach that's emerged for verifying requirements.

The essential idea behind model checking is A model-checking tool accepts system requirements or design (called models) and a property(called specification) that the final system is expected to satisfy.

The tool then outputs yes if the given model satisfies given specifications and generates a counterexample otherwise. The counterexample details why the model doesn't satisfy the specification. By studying the counterexample, you can pinpoint the source of the error in the model, correct the model, and try again. The idea is that by ensuring that the model satisfies enough system properties, we increase our confidence in the correctness of the model. The system requirements are called models because they represent requirements or design.

www.EnggTree.com

So what formal language works for defining models? There's no single answer, since requirements (or design) for systems in different application domains vary greatly.

For instance, requirements of a banking system and an aerospace system differ in size, structure, complexity, nature of system data, and operations performed.

In contrast, most real-time embedded or safety-critical systems are control-oriented rather than data-oriented—meaning that dynamic behavior is much more important than business logic (the structure of and operations on the internal data maintained by the system).

Such control-oriented systems occur in a wide variety of domains: aerospace, avionics, automotive, biomedical

Downloaded from EnggTree.com

EnggTree.com

instrumentation, industrial automation and process control, railways, nuclear power plants, and so forth. Even communication and security protocols in digital hardware systems can be thought of as control oriented.

For control-oriented systems, finite state machines (FSM) are widelyaccepted as a good, clean, and abstract notation for defining

www.EnggTree.com

Downloaded from EnggTree.com

EnggTree.com

requirements and design. But of course, a “pure” FSM is not adequate for modeling complex real-life industrial systems. We also need to:

be able to modularize the requirements to view them at different levels of detail

- have a way to combine requirements (or design) of components
- be able to state variables and facilities to update them in order to use them in guards on transitions.

In short, we need extended finite state machines (EFSM) . Most model checking tools have their own rigorous formal language for defining models, but most of them are some variant of the EFSM.

www.EnggTree.com

Downloaded from EnggTree.com

EnggTree.com

1. SOFTWARE PROJECT MANAGEMENT

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. This article focuses on discussing Software Project Management (SPM).

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. Most of the software products are made to fit clients' requirements. The most important is that basic technology changes and advances so frequently and rapidly that the experience of one product may not be applied to the other one.

Such types of business and environmental constraints increase risk in software development hence 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.

www.EnggTree.com

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 goalof 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.

Downloaded from EnggTree.com

EnggTree.com

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 andtracking 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 existingservices.

www.EnggTree.com

SOFTWARE CONFIGURATION MANAGEMENT

Whenever software is built, there is always scope for improvement and those improvements bring picture changes. Changes may be required to modify or update any existing solution or to create a new solution for a problem.

Requirements keep on changing daily so we need to keep on upgrading our systems based on the current requirements and needs to meet desired outputs. Changes should be analyzed before they are made to the existing system, recorded before they are implemented, reported to have details of before and after, and controlled in a manner that will improve quality and reduce error.

This is where the need for System Configuration Management comes. **System Configuration Management (SCM)** isan arrangement of exercises that controls change by recognizing the items for change, setting up connections between those things, making/characterizing instruments for

Downloaded from EnggTree.com

EnggTree.com

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.

www.EnggTree.com

Downloaded from EnggTree.com

EnggTree.com

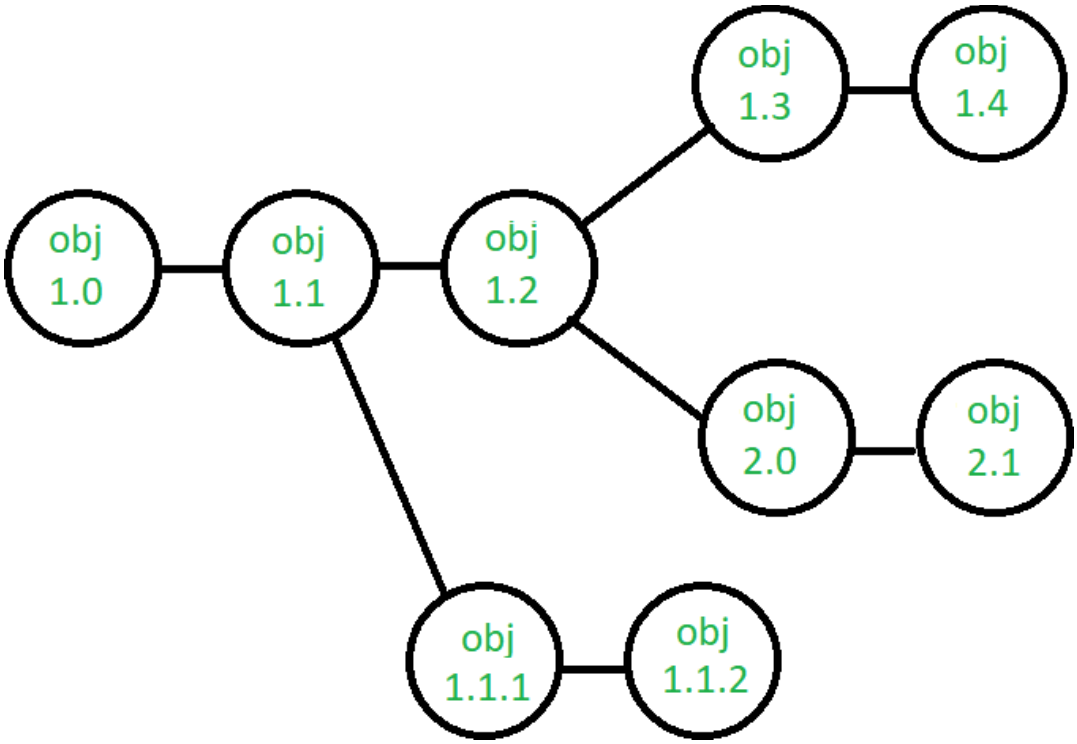
Processes involved in SCM – Configuration management provides a disciplined environment for smooth control of work products.
It involves the following activities:

1. **Identification and Establishment –** Identifying the configuration items from products that compose baselines atgiven 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:

www.EnggTree.com

Downloaded from EnggTree.com

EnggTree.com

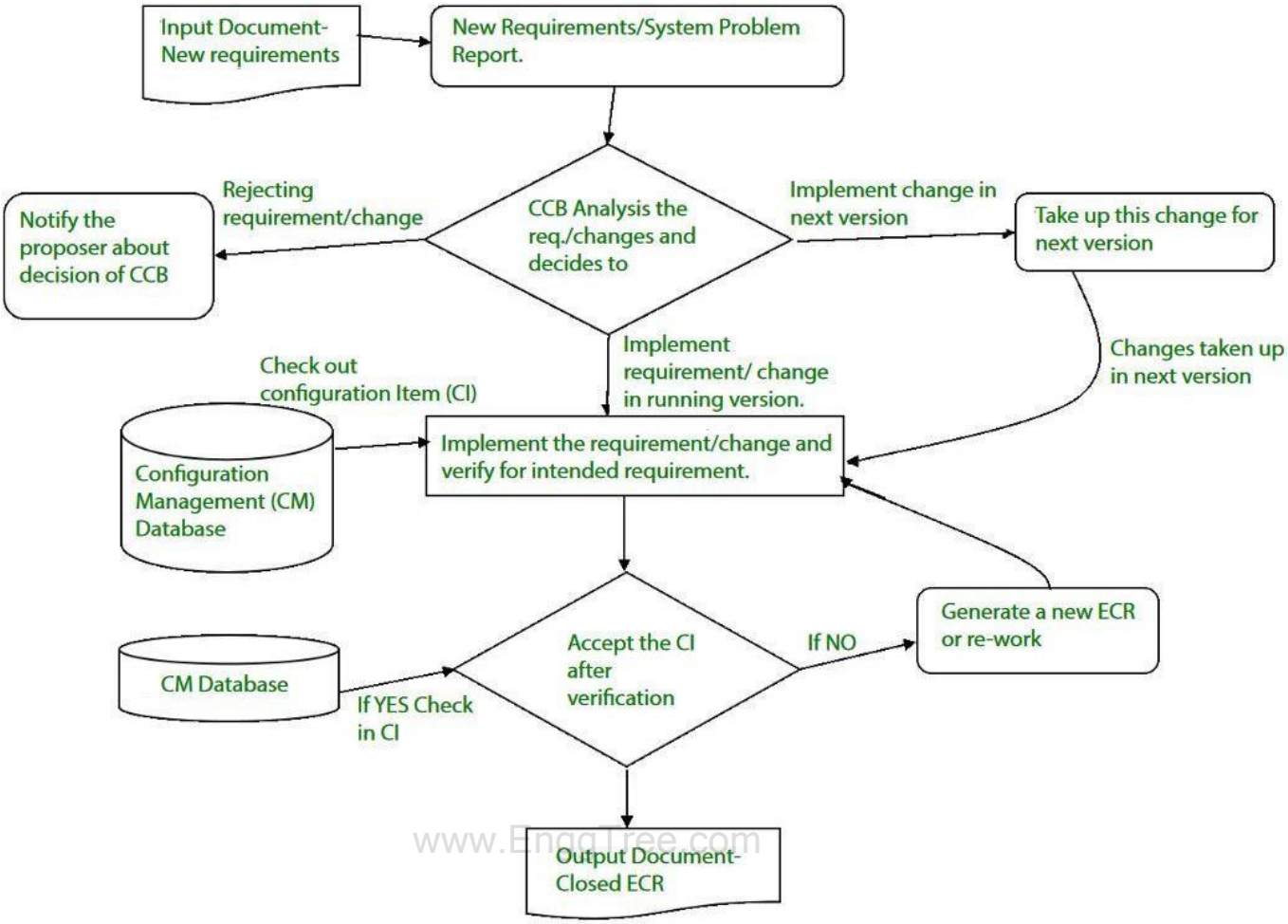


Suppose after some changes, the version of the configuration object changes from 1.0 to 1.1. Minor corrections and changes result in versions 1.1.1 and 1.1.2, which is followed by a major update that is object 1.2. The development of object 1.0 continues through 1.3 and 1.4, but finally, a noteworthy change to the object results in a new evolutionary path, version 2.0. Both versions are currently supported.

3. **Change control –** Controlling changes to Configurationitems (CI). The change control process is explained in Figure below:

Downloaded from EnggTree.com

EnggTree.com



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.

Downloaded from EnggTree.com

EnggTree.com

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 ofitems in the SCM system and tracks action items from the audit to closure.

4. **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 weremade, and manage the integration of these changes into the final product 0 seconds of 17 seconds Volume 0%

www.EnggTree.com

Importance of Software Configuration Management

1. Effective Bug Tracking: Linking code modifications to issues that have been reported, makes bug tracking moreeffective.
2. Continuous Deployment and Integration: SCM combines with continuous processes to automate deployment and testing, resulting in more dependable and timely softwaredelivery.
3. Risk management: SCM lowers the chance of introducingcritical flaws by assisting in the early detection and correction of problems.

Downloaded from EnggTree.com

EnggTree.com

4. Support for Big Projects: Source Code Control (SCM) offersan 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.

Why need for System configuration management?

1. Replicability: Software version control (SCM) makes ensures that a software system can be replicated at any stage of its development. This is necessary for testing, debugging, and upholding consistent environments in production, testing, and development.
2. Identification of Configuration: Source code, documentation, and executable files are examples of configuration elements that SCM helps in locating and labeling. The management of a system's constituent parts and their interactions depend on this identification.
3. Effective Process of Development: By automating monotonous processes like managing dependencies, merging changes, and resolving disputes, SCM simplifies the development process. Error risk is decreased and efficiency is increased because of this automation.

Key objectives of SCM

1. **Control the evolution of software systems:** SCM helps to ensure that changes to a software system are properly planned, tested, and integrated into the final product.
2. **Enable collaboration and coordination:** SCM helps teams to collaborate and coordinate their work, ensuring that changes are properly integrated and that everyone is working from the same version of the software system.
3. **Provide version control:** SCM provides version control for software systems, enabling teams to manage and track different

Downloaded from EnggTree.com

EnggTree.com

versions of the system and to revert to earlier versions if necessary.

www.EnggTree.com

Downloaded from EnggTree.com

EnggTree.com

4. **Facilitate replication and distribution:** SCM helps to ensure that software systems can be easily replicated and distributed to other environments, such as test, production,and customer sites.
5. SCM is a critical component of software development, and effective SCM practices can help to improve the quality and reliability of software systems, as well as increase efficiencyand reduce the risk of errors.

The main advantages of SCM

1. Improved productivity and efficiency by reducing the timeand effort required to manage software changes.
2. Reduced risk of errors and defects by ensuring that allchanges 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 byensuring 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 allchanges are properly integrated.
3. Potential for conflicts and delays, particularly in large development teams with multiple contributors.

www.EnggTree.com

PROJECT SCHEDULING

Project-task scheduling is a significant project planning activity. It comprises deciding which functions would be taken up when. To schedule the project plan, a software project manager wants to do the following:

1. Identify all the functions required to complete the project.
2. Break down large functions into small activities.

Downloaded from EnggTree.com

EnggTree.com

3. Determine the dependency among various activities.
4. Establish the most likely size for the time duration required tocomplete the activities.
5. Allocate resources to activities.
6. Plan the beginning and ending dates for different activities.
7. Determine the critical path. A critical way is the group of activities that decide the duration of the project.

The first method in scheduling a software plan involves identifying allthe functions required to complete the project. A good judgment of theintricacies of the project and the development process helps the supervisor to identify the critical role of the project effectively.

Next, the large functions are broken down into a valid set of small activities which would be assigned to various engineers. The work breakdown structure formalism supports the manager to breakdown the function systematically after the project manager has broken down the purpose and constructs the work breakdown structure; he has to find the dependency among the activities.

www.EnggTree.com

Dependency among the various activities determines the order in which the various events would be carried out. If an activity A necessary the results of another activity B,then activity A must be scheduled after activity B. In general, the function dependencies describe a partial ordering among functions, i.e., each service may precede a subset of other functions, but some functions might not have any precedence ordering describe between them (called concurrent function). The dependency among the activities is defined in the pattern of an activity network.

Once the activity network representation has been processed out, resources are allocated to every activity. Resource allocation is usually done using a Gantt chart. After resource allocation is completed, a PERT chart representation is developed.

The PERT chart representation is useful for program monitoring and control. For task scheduling, the project plan needs to decompose the project functions into a set of activities. The time frame when every activity is to be performed is to be determined. The end of every action is called a milestone.

Downloaded from EnggTree.com

EnggTree.com

The project manager tracks the function of a project by audit the timely completion of the milestones. If he examines that the

www.EnggTree.com

Downloaded from EnggTree.com

EnggTree.com

Milestones start getting delayed, and then he has to handle the activities Carefully so that the complete deadline can still be met.

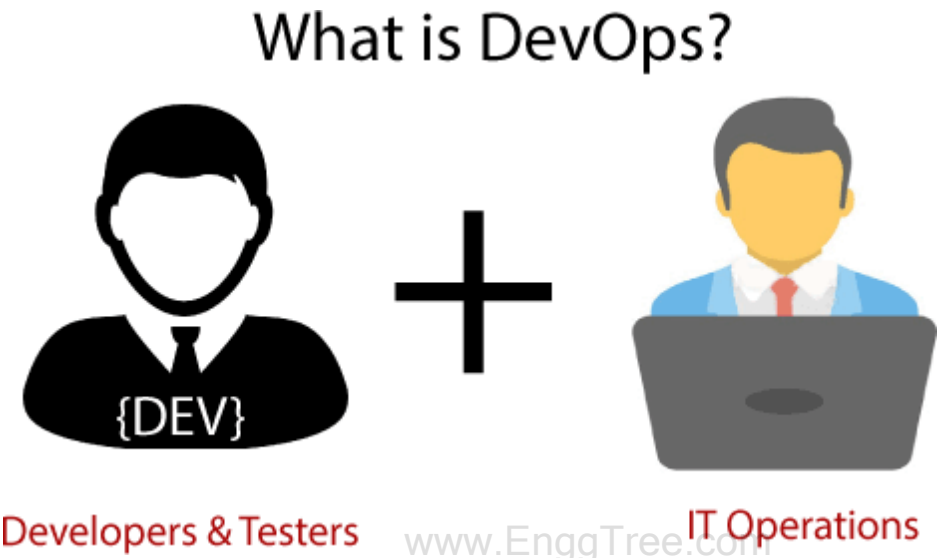
www.EnggTree.com

Downloaded from EnggTree.com

DEVOPS: MOTIVATION

DevOps : Motivation

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



www.EnggTree.com

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.

Backward Skip 10sPlay VideoForward Skip 10s

ADVERTISEMENT

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.

DevOps can also be defined as a sequence of development and IT operations with better communication and collaboration.

Downloaded from EnggTree.com

EnggTree.com

DevOps has become one of the most valuable business disciplines for enterprises or organizations. With the help of DevOps, **quality**, and **speed** of the application delivery has improved to a great extent.

DevOps is nothing but a practice or methodology of making "**Developers**" and "**Operations**" folks work together. DevOps represents a change in the IT culture with a complete focus on rapid IT service delivery through the adoption of agile practices in the context of a system-oriented approach.

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

CLOUD AS A PLATFORM

There are a ton 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.

www.EnggTree.com

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 centres 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:

ADVERTISEMENT

- **Public Cloud**: Third-party providers that distribute computing services over the Internet are known as public cloud platforms. A

Downloaded from EnggTree.com

EnggTree.com

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.

Top 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:

www.EnggTree.com

Cost

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

Global scale

ADVERTISEMENT

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.

Downloaded from EnggTree.com

EnggTree.com

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

www.EnggTree.com

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.

OPERATIONS

1. IAAS: Infrastructure As A Service (IAAS) is means of delivering computing infrastructure as on-demand services. It is one of the three fundamental cloud service models. The user purchases servers, software data center space, or network equipment and rent those resources through a fully outsourced, on-demand service model. It allows dynamic scaling and the resources are distributed as a service. It generally includes multiple-user on a single piece of hardware. It totally depends upon the customer to choose its resources wisely and as per need. Also, it provides billing management too.

2. PAAS: Platform As A Service (PAAS) is a cloud delivery model for applications composed of services managed by a third party. It

Downloaded from EnggTree.com

EnggTree.com

provides elastic scaling of your application which allows developers

www.EnggTree.com

Downloaded from EnggTree.com

to build applications and services over the internet and the deployment models include public, private and hybrid.
Basically, it is a service where a third-party provider provides both software and hardware tools to the cloud computing. The tools which are provided are used by developers. PAAS is also known as Application PAAS. It helps us to organize and maintain useful applications and services. It has a well-equipped management system and is less expensive compared to IAAS.

0 seconds of 17 secondsVolume 0%

3. SAAS: Software As A Service (SAAS) allows users to run existing online applications and it is a model software that is deployed as a hosting service and is accessed over Output Rephrased/Re-written Text the internet or software delivery model during which software and its associated data are hosted centrally and accessed using their client, usually an online browser over the web. SAAS services are used for the development and deployment of modern applications.
It allows software and its functions to be accessed from anywhere with good internet connection device and a browser. An application is hosted centrally and also provides access to multiple users across various locations via the internet.

Difference between IAAS, PAAS and SAAS :

Basis Of	IAAS	PAAS	SAAS
Stands for	Infrastructure as a service.	Platform as a service.	Software as a service.
Uses	IAAS is used by network architects.	PAAS is used by developers.	SAAS is used by the end user.
Access	IAAS gives access to the resources like	PAAS gives access to run time	SAAS gives access to the end user.

Downloaded from EnggTree.com

EnggTree.com

Basis Of	IAAS	PAAS	SAAS
	virtual machines and virtual storage.	environment to deployment and development tools for application.	
Model	It is a service model that provides virtualized computing resources over the internet.	It is a cloud computing model that delivers tools that are used for the development of applications.	It is a service model in cloud computing that hosts software to make it available to clients.
Technical understanding.	It requires technical knowledge.	Some knowledge is required for the basic setup.	There is no requirement about technicalities company handles everything.
Popularity	It is popular among developers and researchers.	It is popular among developers who focus on the development of apps and scripts.	It is popular among consumers and companies, such as file sharing, email, and networking.

www.EnggTree.com

Downloaded from EnggTree.com

Basis Of	IAAS	PAAS	SAAS
Percentage rise	It has around a 12% increment.	It has around 32% increment.	It has about a 27 % rise in the cloud computing model.
Usage	Used by the skilled developer to develop unique applications.	Used by mid-level developers to build applications.	Used among the users of entertainment.
Cloud services.	Amazon Web Services, sun, vCloud Express.	Facebook, and Google search engine.	MS Office web, Facebook and Google Apps.
Enterprise services.	AWS virtual private cloud.	Microsoft Azure.	IBM cloud analysis.
Outsourced cloud services.	Salesforce	Force.com, Gigaspaces.	AWS, Terremark
User Controls	Operating System, Runtime, Middleware, and Application data	Data of the application	Nothing

www.EnggTree.com

Downloaded from EnggTree.com

Basis Of	IAAS	PAAS	SAAS
Others	It is highly scalable and flexible.	It is highly scalable to suit the different businesses according to resources.	It is highly scalable to suit the small, mid and enterprise level business

Advantages of IaaS

- The resources can be deployed by the provider to a customer's environment at any given time.
- Its ability to offer the users to scale the business based on their requirements.
- The provider has various options when deploying resources including virtual machines, applications, storage, and networks.
- It has the potential to handle an immense number of users.
- It is easy to expand and saves a lot of money. Companies can afford the huge costs associated with the implementation of advanced technologies.
- Cloud provides the architecture.
- Enhanced scalability and quite flexible.
- Dynamic workloads are supported.

Disadvantages of IaaS

- Security issues are there.
- Service and Network delays are quite a issue in IaaS.

Advantages of PaaS –

- Programmers need not worry about what specific database or language the application has been programmed in.
- It offers developers the to build applications without the overhead of the underlying operating system or infrastructure.
- Provides the freedom to developers to focus on the application's design while the platform takes care of the language and the database.

Downloaded from EnggTree.com

EnggTree.com

- It is flexible and portable.
- It is quite affordable.
- It manages application development phases in the cloud veryefficiently.

Disadvantages of PaaS

- Data is not secure and is at big risk.
- As data is stored both in local storage and cloud, there arehigh chances of data mismatch while integrating the data.

Advantages of SaaS

- It is a cloud computing service category providing a wide range of hosted capabilities and services. These can be usedto build and deploy web-based software applications.
- It provides a lower cost of ownership than on-premises software. The reason is it does not require the purchase orinstallation of hardware or licenses.
- It can be easily accessed through a browser along a thinclient.
- No cost is required for initial setup.
- Low maintenance costs.
- Installation time is less, so time is managed properly.

Disadvantages of SaaS

- Low performance.
- It has limited customization options.
- It has security and data concerns.

Downloaded from EnggTree.com

EnggTree.com

DEPLOYMENT PIPELINE

In software development, a **deployment pipeline** is a system of automated processes designed to quickly and accurately move new code additions and updates from version control to production. In past development environments, manual steps were necessary when writing, building, and deploying code. This was extremely time consuming for both developers and operations teams, as they were responsible for performing tedious manual tasks such as code testingand code releases.

The introduction of automation in a deployment pipeline allowed development teams to focus more on innovating and

www.EnggTree.com

Downloaded from EnggTree.com

EnggTree.com

improving the end product for the user. By reducing the need for any manual tasks, teams are able to deploy new code updates much quicker and with less risk of any human error.

In this article, we will break down the different stages of a deployment pipeline, how one is built, the benefits of a deploymentpipeline for software development, as well as some helpful tools to get the most out of your system.

Main Stages of a Deployment Pipeline

There are four main stages of a deployment pipeline:

1. Version Control
2. Acceptance Tests
3. Independent Deployment
4. Production Deployment

Version Control is the first stage of the pipeline. This occurs after a developer has completed writing a new code addition and committed it to a source control repository such as GitHub. Once the commit has been made, the deployment pipeline is triggered and the code is automatically compiled, unit tested, analyzed, and run through installer creation. If and when the new code passes this version control stage, binaries are created and stored in an artifact repository. The validated code then is ready for the next stage in the deployment pipeline.

In the **Acceptance Tests** stage of the deployment pipeline, the newly compiled code is put through a series of tests designed to verify the code against your team's predefined acceptance criteria. These tests will need to be custom-written based on your company goals and user expectations for the product. While these tests run automatically once integrated within the deployment pipeline, it's important to be sure toupdate and modify your tests as needed to consistently meet rising user and company expectations.

Downloaded from EnggTree.com

EnggTree.com

Once code is verified after acceptance testing, it reaches the **Independent Deployment** stage where it is automatically deployed to a development environment. The development environment should be identical (or as close as possible) to the production environment in order to ensure an accurate representation for functionality tests. Testing in a development environment allows teams to squash any remaining bugs without affecting the live experience for the user.

The final stage of the deployment pipeline is **Production Deployment**. This stage is similar to what occurs in Independent Deployment, however, this is where code is made live for the user rather than a separate development environment. Any bugs or issues should have been resolved at this point to avoid any negative impact on user experience. DevOps or operations typically handle this stage of the pipeline, with an ultimate goal of zero downtime. Using Blue/Green Drops or Canary Releases allows teams to quickly deploy new updates while allowing for quick version rollbacks in case an unexpected issue does occur.

Benefits of a Deployment Pipeline

www.EnggTree.com

Building a deployment pipeline into your software engineering system offers several advantages for your internal team, stakeholders, and the end user. Some of the primary benefits of an integrated deployment pipeline include:

- Teams are able to release new product updates and features much faster.
- There is less chance of human error by eliminating manual steps.
- Automating the compilation, testing, and deployment of code allows developers and other DevOps team members to focus more on continuously improving and innovating a product.
- Troubleshooting is much faster, and updates can be easily rolled back to a previous working version.
- Production teams can better respond to user wants and needs with faster, more frequent updates by focusing on smaller

Downloaded from EnggTree.com

EnggTree.com

releases as opposed to large, waterfall updates of past production systems.

How to Build a Deployment Pipeline

A company's deployment pipeline must be unique to their company and user needs and expectations, and will vary based on their type of product or service. There is no one-size-fits-all approach to creating a deployment pipeline, as it requires a good amount of upfront planning and creation of tests.

When planning your deployment pipeline, there are three essential components to include:

- **Build Automation (Continuous Integration):** Build automation, also referred to as Continuous Integration or CI for short, are automated steps within development designed for continuous integration – the compilation, building, and mergingof new code.
- **Test Automation:** Test automation relies on the creation of custom-written tests that are automatically triggered throughout a deployment pipeline and work to verify new compiled code
against your organization's predetermined acceptance criteria.
- **Deploy Automation (Continuous Deployment/Delivery):** Like continuous integration, deploy automation with Continuous Deployment/Delivery (CD for short) helps expedite code delivery by automating the process of releasing code to a shared repository, and then automatically deploying the updates to a development or production environment.

www.EnggTree.com

When building your deployment pipeline, the primary goal should be to eliminate the need for any manual steps or intervention. This meanswriting custom algorithms for automatically compiling/building, testing, and deploying new code from development. By taking these otherwise tedious and repetitive steps off developers and other DevOps team members, they can focus more on creating new,
innovative product updates and features for today's highly
competitive user base.

Downloaded from EnggTree.com

EnggTree.com

What Are Continuous Integration (CI) and Continuous Delivery (CD) Pipelines?

A Continuous Integration (CI) and Continuous Delivery (CD) Pipeline works by continuously compiling, validating, and deploying new code updates as they are written – rather than waiting for specific merge or release days. This allows team to make faster, more frequent updates to a product with improved accuracy from introducing automated steps. CI/CD Pipelines are a key component of an efficier full deployment pipeline.

Deployment Pipeline Tools

Making use of available tools will help to fully automate and get the most out of your deployment pipeline. When first building a deployment pipeline, there are several essential tool categories that must be addressed, including source control, build/compilation, containerization, configuration management, and monitoring.

www.EnggTree.com

A development pipeline should be constantly evolving, improving and introducing new tools to increase speed and automation. Some favorite tools for building an optimal deployment pipeline include:

- Jenkins
- Azure DevOps
- CodeShip
- PagerDuty

DEPLOYMENT TOOLS

DevOps tools make it convenient and easier for companies to reduce the probability of errors and maintain continuous integration in operations. It addresses the key aspects of a company. DevOps tools automate the whole process and automatically build, test, and deploy the features.

DevOps tools make the whole deployment process and easy going

Downloaded from EnggTree.com

EnggTree.com

one and they can help you with the following aspects:

- Increased development.
- Improvement in operational efficiency.
- Faster release.
- Non-stop delivery.
- Quicker rate of innovation.
- Improvement in collaboration.

www.EnggTree.com

Downloaded from EnggTree.com

- Seamless flow in the process chain.

The DevOps tools play a very vital role in every organization, all of which are discussed here. Read along to know all about it.

DEPLOYMENT TOOLS

DevOps tools make it convenient and easier for companies to reduce the probability of errors and maintain continuous integration in operations. It addresses the key aspects of a company. DevOps tools automate the whole process and automatically build, test, and deploy the features.

DevOps tools make the whole deployment process and easy going one and they can help you with the following aspects:

- Increased development.
- Improvement in operational efficiency.
- Faster release.
- Non-stop delivery.
- Quicker rate of innovation.
- Improvement in collaboration.

www.EnggTree.com

Downloaded from EnggTree.com

EnggTree.com

- Seamless flow in the process chain.

The DevOps tools play a very vital role in every organization, all of which are discussed here. Read along to know all about it.

Architecture

The following diagram shows the flow of data in a deployment pipeline. It illustrates how you can turn your artifacts into resources.

Deployment pipelines are often part of a larger continuous integration/continuous deployment (CI/CD) workflow and are typically implemented using one of the following models:

- **Push model:** In this model, you implement the deployment pipeline using a central CI/CD system such as Jenkins or GitLab. This CI/CD system might run on Google Cloud, on-premises, or on a different cloud environment. Often, the same CI/CD system is used to manage multiple deployment pipelines.

  The push model leads to a centralized architecture with a few CI/CD systems that are used for managing a potentially large number of resources or applications. For example, you might use a single Jenkins or GitLab instance to manage your entire production environment, including all its projects and applications.
- **Pull model:** In this model, the deployment process is implemented by an agent that is deployed alongside the resource–for example, in the same Kubernetes cluster. The agent pulls artifacts or source code from a centralized location, and deploys them locally. Each agent manages one or two resources.

  The pull model leads to a more decentralized architecture with a potentially large number of single-purpose agents.

  Compared to manual deployments, consistently using deployment pipelines can have the following benefits:

- Increased efficiency, because no manual work is required.
- Increased reliability, because the process is fully automated and repeatable.

Downloaded from EnggTree.com

- Increased traceability, because you can trace all deployments to changes in code or to input artifacts.

To perform, a deployment pipeline requires access to the resources it manages:

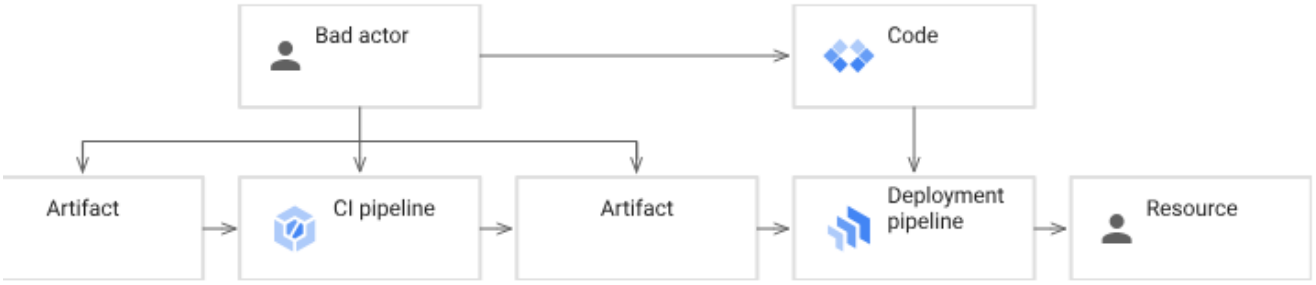
- A pipeline that deploys infrastructure by using tools like Terraform might need to create, modify, or even delete resources like VM instances, subnets, or Cloud Storage buckets.
- A pipeline that deploys applications might need to upload new container images to Artifact Registry, and deploy new application versions to App Engine, Cloud Run, or Google Kubernetes Engine (GKE).
- A pipeline that manages settings or deploys configuration files might need to modify VM instance metadata, Kubernetes configurations, or modify data in Cloud Storage.

If your deployment pipelines aren't properly secured, their access to Google Cloud resources can become a weak spot in your security posture. Weakened security can lead to several kinds of attacks, including the following:

- **Pipeline poisoning attacks:** Instead of attacking a resource directly, a bad actor might attempt to compromise the deployment pipeline, its configuration, or its underlying infrastructure. Taking advantage of the pipeline's access to Google Cloud, the bad actor could make the pipeline perform malicious actions on Cloud resources, as shown in the following diagram:

- **Supply chain attacks:** Instead of attacking the deployment pipeline, a bad actor might attempt to compromise or replace pipeline input—including source code, libraries, or container images, as shown in the following diagram:

Downloaded from EnggTree.com



To determine whether your deployment pipelines are appropriately secured, it's insufficient to look only at the allow policies and deny policies of Google Cloud resources in isolation. Instead, you must consider the entire graph of systems that directly or indirectly grant access to a resource. This graph includes the following information:

- The deployment pipeline, its underlying CI/CD system, and its underlying infrastructure
- The source code repository, its underlying servers, and its underlying infrastructure
- Input artifacts, their storage locations, and their underlying infrastructure
- Systems that produce the input artifacts, and their underlying infrastructure

www.EnggTree.com

Complex input graphs make it difficult to identify user access to resources and systemic weaknesses.

The following sections describe best practices for designing deployment pipelines in a way that helps you manage the size of the graph, and reduce the risk of lateral movement and supply chain attacks.

Assess security objectives

Your resources on Google Cloud are likely to vary in how sensitive they are. Some resources might be highly sensitive because they're business critical or confidential. Other resources might be less sensitive because they're ephemeral or only intended for testing purposes.

To design a secure deployment pipeline, you must first understand the resources the pipeline needs to access, and how sensitive these resources are. The more sensitive your resources, the more you should focus on securing the pipeline.

Downloaded from EnggTree.com

EnggTree.com

The resources accessed by deployment pipelines might include:

- Applications, such as Cloud Run or App Engine
- Cloud resources, such as VM instances or Cloud Storage buckets

- Data, such as Cloud Storage objects, BigQuery records, or files

Some of these resources might have dependencies on other resources, for example:

- Applications might access data, cloud resources, and other applications.
- Cloud resources, such as VM instances or Cloud Storage buckets, might contain applications or data.

www.EnggTree.com

As shown in the preceding diagram, dependencies affect how sensitive a resource is. For example, if you use an application that accesses highly sensitive data, typically you should treat that application as highly sensitive. Similarly, if a cloud resource like a Cloud Storage bucket contains sensitive data, then you typically should treat the bucket as sensitive.

Because of these dependencies, it's best to first assess the sensitivity of your data. Once you've assessed your data, you can examine the dependency chain and assess the sensitivity of your Cloud resources and applications.

Categorize the sensitivity of your data

To understand the sensitivity of the data in your deployment pipeline, consider the following three objectives:

- **Confidentiality**: You must protect the data from unauthorized access.
- **Integrity**: You must protect the data against unauthorized modification or deletion.

Downloaded from EnggTree.com

EnggTree.com

- **Availability**: You must ensure that authorized people and systems can access the data in your deployment pipeline.

For each of these objectives, ask yourself what would happen if your pipeline was breached:

- **Confidentiality:** How damaging would it be if data was disclosed to a bad actor, or leaked to the public?
- **Integrity:** How damaging would it be if data was modified or deleted by a bad actor?
- **Availability:** How damaging would it be if a bad actor disrupted your data access?

To make the results comparable across resources, it's useful to introduce security categories. Standards for Security Categorization (FIPS-199) suggests using the following four categories:

- **High:** Damage would be severe or catastrophic
- **Moderate:** Damage would be serious
- **Low:** Damage would be limited
- **Not applicable:** The standard doesn't apply

www.EnggTree.com

Depending on your environment and context, a different set of categories could be more appropriate.

The confidentiality and integrity of pipeline data exist on a spectrum, based on the security categories just discussed. The following subsections contain examples of resources with different confidentiality and integrity measurements:

Resources with low confidentiality, but low, moderate, and high integrity

The following resource examples all have low confidentiality:

- **Low integrity:** Test data
- **Moderate integrity:** Public web server content, policy constraints for your organization

Downloaded from EnggTree.com

EnggTree.com

- **High integrity:** Container images, disk images, application configurations, access policies (allow and deny lists), liens, access-level data

Resources with medium confidentiality, but low, moderate, and high integrity

The following resource examples all have medium confidentiality:

- **Low integrity:** Internal web server content
- **Moderate integrity:** Audit logs
- **High integrity:** Application configuration files

Resources with high confidentiality, but low, moderate, and high integrity

The following resource examples all have high confidentiality:

- **Low integrity:** Usage data and personally identifiable information
- **Moderate integrity:** Secrets
- **High integrity:** Financial data, KMS keys

www.EnggTree.com

Categorize applications based on the data that they access

When an application accesses sensitive data, the application and the deployment pipeline that manages the application can also become sensitive. To qualify that sensitivity, look at the data that the application and the pipeline need to access.

Once you've identified and categorized all data accessed by an application, you can use the following categories to initially categorize the application before you design a secure deployment pipeline:

- **Confidentiality**: Highest category of any data accessed
- **Integrity**: Highest category of any data accessed
- **Availability**: Highest category of any data accessed

This initial assessment provides guidance, but there might be additional factors to consider—for example:

Downloaded from EnggTree.com

- Two sets of data might have low-confidentiality in isolation. But when combined, they could reveal new insights. If an application has access to both sets of data, you might need to categorize it as medium- or high-confidentiality.
- If an application has access to high-integrity data, then you should typically categorize the application as high-integrity. But if that access is read only, a categorization of high-integrity might be too strict.

For details on a formalized approach to categorize applications, see Guide for Mapping Types of Information and Information Systems to Security Categories (NIST SP 800-60 Vol. 2 Rev1).

Categorize cloud resources based on the data and applications they host

Any data or application that you deploy on Google Cloud is hosted by a Google Cloud resource:

- An application might be hosted by an App Engine service, a VM instance, or a GKE cluster.
- Your data might be hosted by a persistent disk, a Cloud Storage bucket, or a BigQuery dataset.

www.EnggTree.com

When a cloud resource hosts sensitive data or applications, the resource and the deployment pipeline that manages the resource can also become sensitive. For example, you should consider a Cloud Run service and its deployment pipeline to be as sensitive as the application that it's hosting.

After categorizing your data and your applications, create an initial security category for the application. To do so, determine a level from the following categories:

- **Confidentiality**: Highest category of any data or application hosted
- **Integrity**: Highest category of any data or application hosted
- **Availability**: Highest category of any data or application hosted

When making your initial assessment, don't be too strict—for example:

- If you encrypt highly confidential data, treat the encryption key as highly confidential. But, you can use a lower security category for the resource containing the data.

Downloaded from EnggTree.com

EnggTree.com

- If you store redundant copies of data, or run redundant instances of the same applications across multiple resources, you can make the category of the resource lower than the category of the data or application it hosts.

Constrain the use of deployment pipelines

If your deployment pipeline needs to access sensitive Google Cloud resources, you must consider its security posture. The more sensitive the resources, the better you need to attempt to secure the pipeline. However, you might encounter the following practical limitations:

- When using existing infrastructure or an existing CI/CD system, that infrastructure might constrain the security level you can realistically achieve. For example, your CI/CD system might only support a limited set of security controls, or it might be running on infrastructure that you consider less secure than some of your production environments.
- When setting up new infrastructure and systems to run your deployment pipeline, securing all components in a way that meets your most stringent security requirements might not be cost effective.

www.EnggTree.com

To deal with these limitations, it can be useful to set constraints on what scenarios should and shouldn't use deployment pipelines and a particular CI/CD system. For example, the most sensitive deployments are often better handled outside of a deployment pipeline. These deployments could be manual, using a privileged session management system or a privileged access management system, or something else, like tool proxies.

To set your constraints, define which access controls you want to enforce based on your resource categories. Consider the guidance offered in the following table:

Category of resource	**Access controls**
Low	No approval required
Moderate	Team lead must approve
High	Multiple leads must approve and actions must be recorded

Downloaded from EnggTree.com

EnggTree.com

Contrast these requirements with the capabilities of your source code management (SCM) and CI/CD systems by asking the following questions and others:

- Do your SCM or CI/CD systems support necessary access controls and approval mechanisms?
- Are the controls protected from being subverted if bad actors attack the underlying infrastructure?
- Is the configuration that defines the controls appropriately secured?

Depending on the capabilities and limitations imposed by your SCM or CI/CD systems, you can then define your data and application constraints for your deployment pipelines. Consider the guidance offered in the following table:

Category of resource	Constraints
Low	Deployment pipelines can be used, and developers can self-approve deployments.
Moderate	Deployment pipelines can be used, but a team lead has to approve every commit and deployment.
High	Don't use deployment pipelines. Instead, administrators have to use a privileged access management system and session recording.

www.EnggTree.com

Maintain resource availability

Using a deployment pipeline to manage resources can impact the availability of those resources and can introduce new risks:

- **Causing outages:** A deployment pipeline might push faulty code or configuration files, causing a previously working system to break, or data to become unusable.
- **Prolonging outages:** To fix an outage, you might need to rerun a deployment pipeline. If the deployment pipeline is broken or unavailable for other reasons, that could prolong the outage.

A pipeline that can cause or prolong outages poses a denial of service risk: A bad actor might use the deployment pipeline to intentionally cause an outage.

Create emergency access procedures

Downloaded from EnggTree.com

EnggTree.com

When a deployment pipeline is the only way to deploy or configure an application or resource, pipeline availability can become critical. In extreme cases, where a deployment pipeline is the only way to manage a business-critical application, you might also need to consider the deployment pipeline business-critical.

Because deployment pipelines are often made from multiple systems and tools, maintaining a high level of availability can be difficult or uneconomical.

You can reduce the influence of deployment pipelines on availability by creating emergency access procedures. For example, create an alternative access path that can be used if the deployment pipeline isn't operational.

Creating an emergency access procedure typically requires most of the following processes:

- Maintain one of more user accounts with privileged access to relevant Google Cloud resources.
- Store the credentials of emergency-access user accounts in a safe location, or use a privileged access management system to broker access.
- Establish a procedure that authorized employees can follow to access the credentials.
- Audit and review the use of emergency-access user accounts.

www.EnggTree.com

Ensure that input artifacts meet your availability demands

Deployment pipelines typically need to download source code from a central source code repository before they can perform a deployment. If the source code repository isn't available, running the deployment pipeline is likely to fail.

Many deployment pipelines also depend on third-party artifacts. Such artifacts might include libraries from sources such as npm, Maven Central, or the NuGet Gallery, as well as container base images, and .deb, and .rpm packages. If one of the third-party sources is unavailable, running the deployment pipeline might fail.

To maintain a certain level of availability, you must ensure that the input artifacts of your deployment pipeline all meet the same or higher availability requirements. The following list can help you ensure the availability of input artifacts:

Downloaded from EnggTree.com

EnggTree.com

- Limit the number of sources for input artifacts, particularly third-party sources
- Maintain a cache of input artifacts that deployment pipelines can use if source systems are unavailable

Treat deployment pipelines and their infrastructure like production systems

Deployment pipelines often serve as the connective tissue between development, staging, and production environments. Depending on the environment, they might implement multiple stages:

- In the first stage, the deployment pipeline updates a development environment.
- In the next stage, the deployment pipeline updates a staging environment.
- In the final stage, the deployment pipeline updates the production environment.

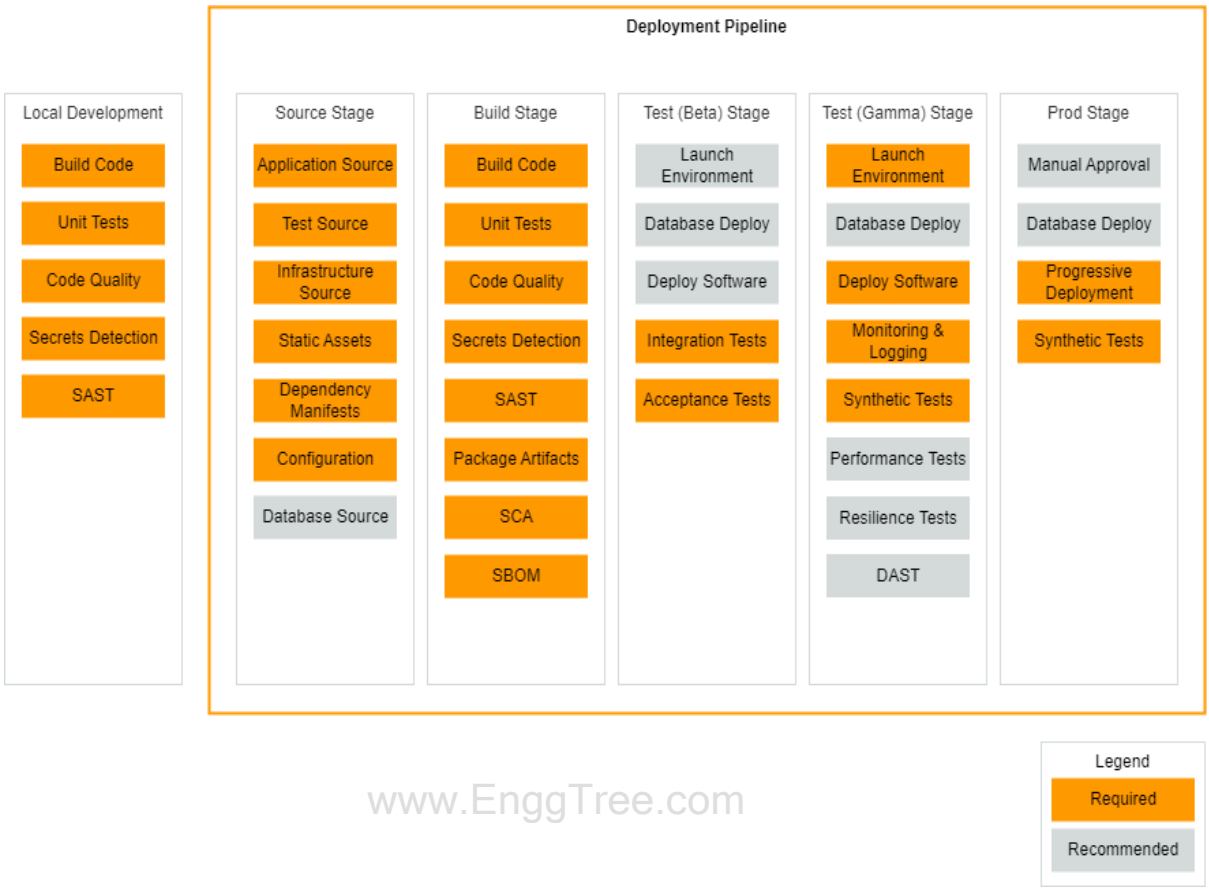
When using a deployment pipeline across multiple environments, ensure that the pipeline meets the availability demands of each environment. Because production environments typically have the highest availability demands, you should treat the deployment pipeline and its underlying infrastructure like a production system. In other words, apply the same access control, security, and quality standards to the infrastructure running your deployment pipelines as you do for your production systems.

www.EnggTree.com

OVERALL ARCHITECTURE BUILDING AND TESTING

Downloaded from EnggTree.com

EnggTree.com



www.EnggTree.com

Downloaded from EnggTree.com