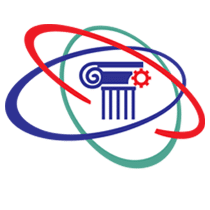
**ACROPOLIS INSTITUTE OF TECHNOLOGY & RESEARCH**



A Software Enginerring Project

**VISUALISER**

Submitted in Partial Fulfillment for the award of

Bachelor of Technology

In

Computer Science

( 2019 – 20 )

# Submitted By:

- Gourav Jaiswal (0827CS181078)

- Himanshu Pandey (0827CS181092)

- Gourav Kadam (0827CS181079)

- Harshit Goyal (0827CS181088)

Problem Statement

|  |  |
| --- | --- |
|  |  |
|  |  | Students have difficulties in understanding Graphs and various Sorting algorithms. various sorting techniques have many linked methodologies which can be remembered but the concepts get faded in the memory after some time. Same thing happens with the concepts of graphs, also graphs require precision, the more precise graphs are made the more accurate results are formed. Manually such accuracy cannot be achieved at all the points.  Solution  Fading of once learned concepts can be forgotten if things are learned visually.  When we speak of graphs and sorting methods, both require precision and good  Visualization . VISUALISER enables the students to learn the concepts of both  graphs and sorting methods by visualizing them instead of mugging up the process. |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |



Software Engineering Approach



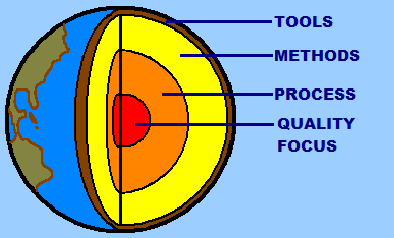
# Introduction

"Paradigm", a Greek word meaning "example", is commonly used to refer to a category of entities that share a common characteristic. Software engineering paradigms are also known as Software engineering models or Software Development Models.

In order to reduce the potential chaos of developing software applications and systems, we use software process models and paradigms that describe the tasks that are required for the building of high-quality software systems. The specific process model or paradigms used to develop a given system depends heavily on the nature of the target system. Use of software paradigms in the development of the software processes has many benefits, including supporting systematic approach and the use of standard approaches and methodologies.

The software engineering paradigm which is also referred to as a software process model or Software Development Life Cycle (SDLC) model is the development strategy that encompasses the process, methods and tools. SDLC describes the period of time that starts with the software system being conceptualized and ends with the software system been discarded after usage.

Four Layers of Software Engineering -

****

**1. A quality Focus :-**

Any engineering approach must rest on an quality. The "Bed Rock" that supports software Engineering is Quality.

**2. Process :-**

Foundation for SE is the Process Layer SE process is the glue that holds all the technology layers together and enables the timely development of computer software.

It forms the base for management control of software project.

**3. Methods :-**

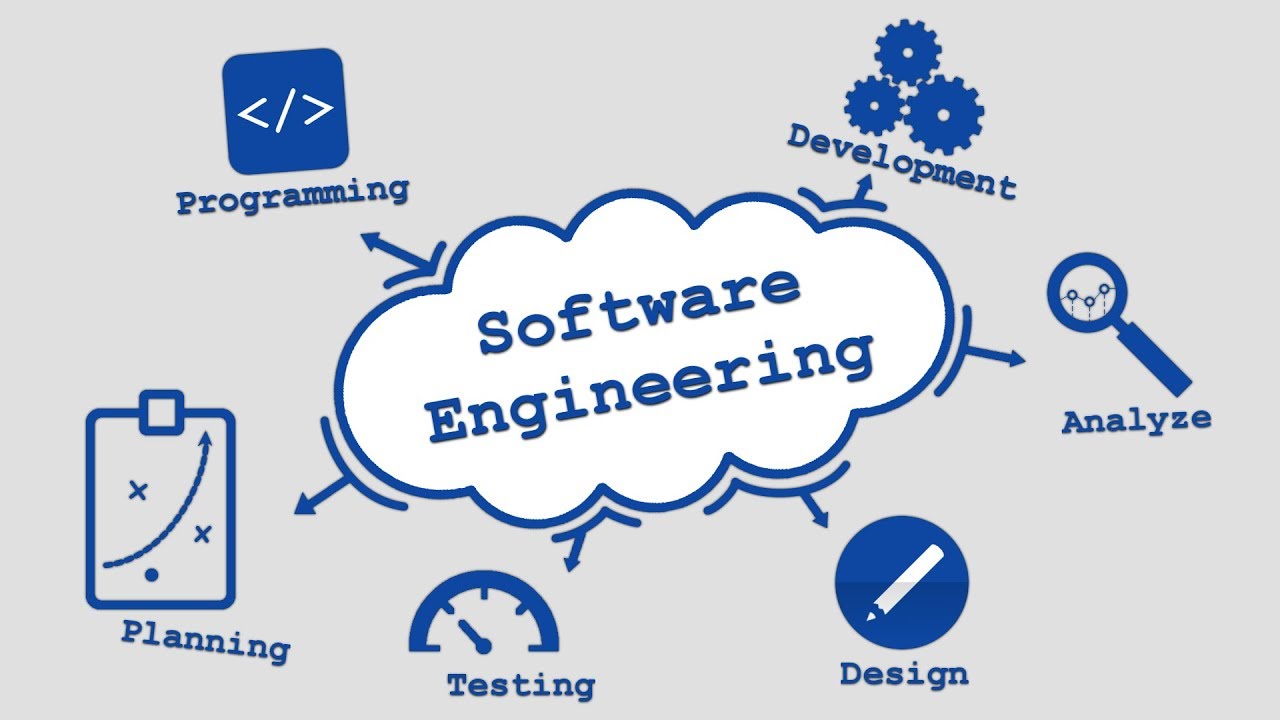
SE methods provide the "Technical Questions" for building Software. Methods contain a broad array of tasks that include communication requirement analysis, design modeling, program construction testing and support.

**4. Tools :-**

SE tools provide automated or semi-automated support for the "Process" and the "Methods". Tools are combined and interrelated so that information created by one tool can be used by another.

Software Engineering Paradigm Applied-

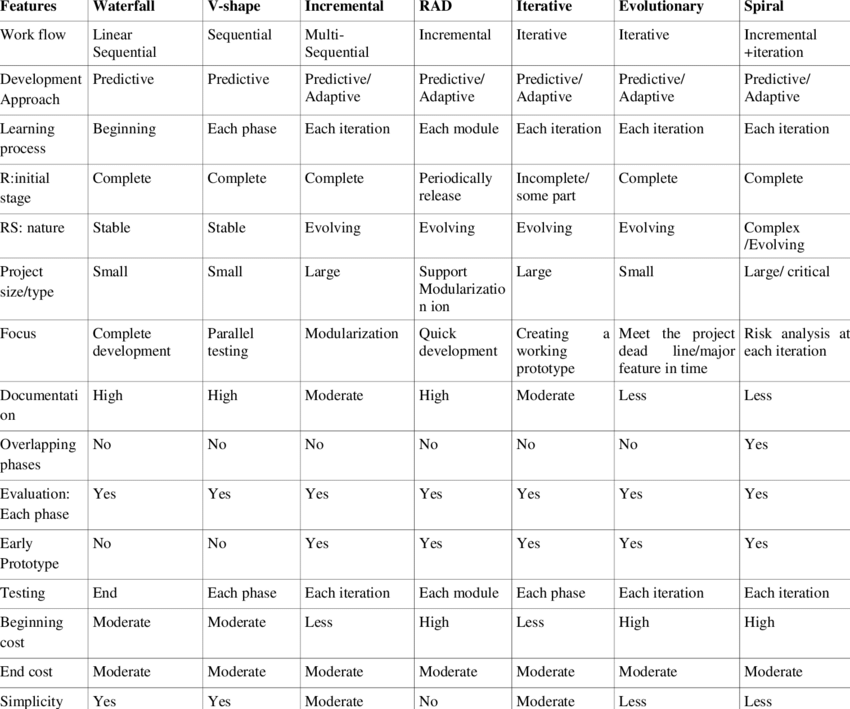
Software Engineering is the establishment and use of sound engineering principles in order to obtain economically software that is both reliable and works efficiently on real systems. To solve actual problems in an industry or organization setting, such as Central or State Govt. Sectors, Public and Private Sectors, Colleges, Schools, etc.



Twelve Principles of Agile Manifesto

* **Customer Satisfaction** − Highest priority is given to satisfy the requirements of customers through early and continuous delivery of valuable software.
* **Welcome Change** − Changes are inevitable during software development. Ever-changing requirements should be welcome, even late in the development phase. Agile processes should work to increase customers' competitive advantage.
* **Deliver a Working Software** − Deliver a working software frequently, ranging from a few weeks to a few months, considering shorter time-scale.
* **Collaboration** − Business people and developers must work together during the entire life of a project.
* **Motivation** − Projects should be built around motivated individuals. Provide an environment to support individual team members and trust them so as to make them feel responsible to get the job done.
* **Face-to-face Conversation** − Face-to-face conversation is the most efficient and effective method of conveying information to and within a development team.
* **Measure the Progress as per the Working Software** − Working software is the key and it should be the primary measure of progress.
* **Maintain Constant Pace** − Agile processes aim towards sustainable development. The business, the developers, and the users should be able to maintain a constant pace with the project.
* **Monitoring** − Pay regular attention to technical excellence and good design to enhance agility.
* **Simplicity** − Keep things simple and use simple terms to measure the work that is not completed.
* **Self-organized Teams** − An agile team should be self-organized and should not depend heavily on other teams because the best architectures, requirements, and designs emerge from self-organized teams.
* **Review the Work Regularly** − Review the work done at regular intervals so that the team can reflect on how to become more effective and adjust its behavior accordingly.

Comparing All Software Devlopment Approachs

****

Prototyping Approach

Definition  
“Prototyping Approach is the development of a working model which may then be developed further into a fully functioning solution".( Wilson .C, (2001) *Software Design and Development: The Preliminary Course.*)  
  
Information  
Prototyping is an engineering technique that has been transferred to computer systems development. The developer makes a small scale model of the proposed program so that users can give feedback and ensure it meets their needs.  
  
Characteristics

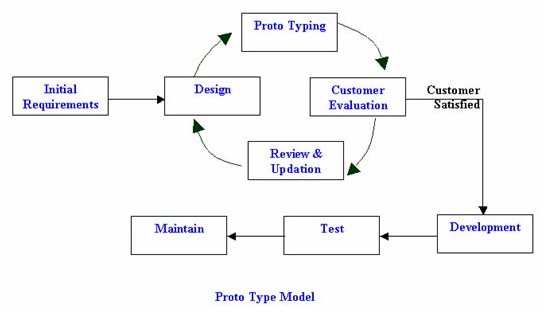
* Prototypes focus on what the users interacts with, the input and output requirements only
* Prototypes are not interested in the code behind the interfaces
* It is a non-formal approach
* It encourages end user participation
* It improves communication between programmer/s and end user/s
* The developer can develop typical screens and then quickly modify them after user feedback
* They give immediate feedback, because the changes can be completed quickly and/or in situ
* You can quickly create the look and feel for the proposed/final product in comparison to the other approaches. Mind you, the final product might take some time to build.
* They typically results in a lower required budget and shorter development time
* They are a good approach for small scale projects
* They can be incorporated within the structured approach
* Prototypes ignore program controls, control structures and error handling because they focus on the user interface.
* Prototypes can evolve into the final product.

# Use

* multimedia
* website's
* online enquiry

# Personnel

* analysis
* programming team



# Reason to Use Prototype

Prototype model is used when the desired system needs to have a lot interaction with the end users. Typically, online ,systems, web interfaces have a very high amount of interaction with end users, are best suited for Prototype model. It might take awhile for a system to be built that allow ease of use and needs minimal training for the end users. Prototyping ensures that the end users constantly work with the system and provide a feedback which is incorporated in the prototype to result in useable system. They are excellent for designing good human computer interface systems. This is the reason why we have used this model in our project.

# Advantages :

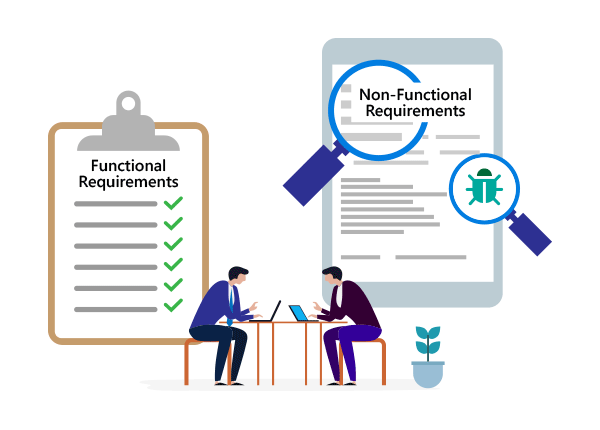
* Here, are important pros/benefits of using Prototyping models:
* Users are actively involved in development. Therefore, errors can be detected in the initial stage of the software development process.
* Missing functionality can be identified, which helps to reduce the risk of failure as Prototyping is also considered as a risk reduction activity.
* Helps team member to communicate effectively
* Customer satisfaction exists because the customer can feel the product at a very early stage.
* There will be hardly any chance of software rejection.
* Quicker user feedback helps you to achieve better software development solutions.
* Allows the client to compare if the software code matches the software specification.
* It helps you to find out the missing functionality in the system.
* It also identifies the complex or difficult functions.
* Encourages innovation and flexible designing.

# Disadvantages:

* Here, are important cons/drawbacks of prototyping model:
* Prototyping is a slow and time taking process.
* The cost of developing a prototype is a total waste as the prototype is ultimately thrown away.
* Prototyping may encourage excessive change requests.
* Some times customers may not be willing to participate in the iteration cycle for the longer time duration.
* There may be far too many variations in software requirements when each time the prototype is evaluated by the customer.
* Poor documentation because the requirements of the customers are changing.
* It is very difficult for software developers to accommodate all the changes demanded by the clients.
* After seeing an early prototype model, the customers may think that the actual product will be delivered to him soon.
* The client may lose interest in the final product when he or she is not happy with the initial prototype.
* Developers who want to build prototypes quickly may end up building sub-standard development solutions.

Functional Requirements

The requirements that specify the functional aspects of software are known as functional requirements. Functional requirements change from one project to another. They define the functionalities provided by the systems or components. Assume a hospital management system. It can have several modules such as login module, patient module, doctor module, appointment module, report module and billing module.  The login module should successfully login to the system when the correct username and password is provided. The patient module should save, edit and delete patient details. The doctor module should save, edit and delete doctor details. The appointment module should schedule, reschedule and delete appointments. The report module should generate medical reports. The billing module should generate bills for payment. Those are some functional requirements for a hospital management system.



Non - Functional Requirements

The requirements that are not related to the functional aspect of software fall into the non functional requirements category. They define the expected characteristics of a software. The users can make assumptions about them. Many users are concerned about getting the non-functional requirements right especially for large systems. A hospital management system should have the following non functional requirements. Speed is a considerable requirement. The system should process data within a minimum response time. The system should be secure. The data should be accessible only by the authorized users. It should be easily maintainable. The software should be a working and a useable product. The data should be reliable and available when necessary. Therefore, the hospital management system should have non functional requirements such as performance, security, maintainability, usability, reliability and availability.

VARIOUS SOFTWARE UML DIAGRAMS



UML Diagram:

UML is a way of visualizing a software program using a collection of diagrams. The notation has evolved from the work of Grady Booch, James Rumbaugh, Ivar Jacobson, and the Rational Software Corporation to be used for object-oriented design, but it has since been extended to cover a wider variety of software engineering projects. Today, UML is accepted by the Object Management Group (OMG) as the standard for modeling software development.

USE CASE Diagram:

Use Case diagrams are used to analyze the high level requirements. These requirements are expressed through different use cases. We notice three main components of this UML diagram:

* **Functional requirements** – represented as use cases; a verb describing an action
* **Actors** – they interact with the system; an actor can be a human being, an organization or an internal or external application
* **Relationships** between actors and use cases – represented using straight arrows

A close up of a map

Description automatically generated

# **ER Diagram:**

Entity relationship diagrams are used in software engineering during the planning stages of the software project. They help to identify different system elements and their relationships with each other. It is often used as the basis for data flow diagrams or DFD’s as they are commonly known.

**A close up of a map

Description automatically generated**

# CLASS DIAGRAM:

Class UML diagram is the most common diagram type for software documentation. Since most software being created nowadays is still based on the Objected-oriented programming paradigm, using class diagrams to document the software turns out to be a common-sense solution. This happens because OOP is based on classes and the relations between them.

**A close up of text on a white background

Description automatically generated**

ACTIVITY DIAGRAM:

Activity diagrams are probably the most important UML diagrams for doing business process modelling. In software development, it is generally used to describe the flow of different activities and actions. These can be both sequential and in parallel. They describe the objects used, consumed or produced by an activity and the relationship between the different activities. The entire above are essential in business process modelling.

**A close up of a map

Description automatically generated**

# DATA FLOW DIAGRAM:

A data flow diagram (DFD) is a way of representing a flow of a data of a process or a system (usually an information system). The DFD also provides information about the outputs and inputs of each entity and the process itself.

# LEVEL 0:

**A screenshot of a cell phone

Description automatically generated**

# LEVEL 1:

**A close up of a map

Description automatically generated**

# LEVEL 2:

**A close up of a map

Description automatically generated**

# SEQUENCE DIAGRAM:

As the name suggests, sequence diagrams describe the sequence of messages and interactions that happen between actors and objects. Actors or objects can be active only when needed or when another object wants to communicate with them. All communication is represented in a chronological manner.

**A close up of text on a white background

Description automatically generated**

Implementation

HTML5:



# **What is HTML?**

Hyper Text Mark up Language is a mark up language that provides a description of the structure/layout of your web page. We define this structure by wrapping content in HTML elements.

An HTML element is formed using a tag, which serves as a descriptor for each piece of content on your page. As an example, the <p> tag is used to describe a paragraph HTML element.

Some other examples of HTML elements include:

<h1>: Highest-level heading

<h6>: Lowest-level heading

<img>: An image

<a>: An anchor which creates a hyperlink to things like other HTML pages, files, email addresses, and more

Full lowdown on HTML5 and why it's important for the web.

# What is HTML5?

HTML5 is the latest version of Hypertext Markup Language, the code that describes web pages. It's actually three kinds of code: HTML, which provides the structure; Cascading Style Sheets (CSS), which take care of presentation; and JavaScript, which makes things happen.

# What's so great about HTML5?

HTML5 has been designed to deliver almost everything you'd want to do online without requiring additional software such as browser plugins. It does everything from animation to apps, music to movies, and can also be used to build incredibly complicated applications that run in your browser.

There's more. HTML5 isn't proprietary, so you don't need to pay royalties to use it. It's also cross-platform, which means it doesn't care whether you're using a tablet or a smartphone, a netbook, notebook or ultra-book or a Smart TV: if your browser supports HTML5, it should work flawlessly. Inevitably, it's a bit more complicated than that. More about that in a moment.

# What does HTML5 do?

We've come a long way since HTML could barely handle a simple page layout. HTML5 can be used to write web applications that still work when you're not connected to the net; to tell websites where you are physically located; to handle high definition video; and to deliver extraordinary graphics.

What's important is that HTML's features - such as the aforementioned geo-location, web apps, video and graphics can be used now, provided your browser supports them

CSS:



CSS is used to control the style of a web document in a simple and easy way.

CSS is the acronym for "Cascading Style Sheet". This tutorial covers both the versions CSS1, CSS2 and CSS3, and gives a complete understanding of CSS, starting from its basics to advanced concepts.

# Why to Learn CSS?

Cascading Style Sheets, fondly referred to as CSS, is a simple design language intended to simplify the process of making web pages presentable.

CSS is a MUST for students and working professionals to become a great Software Engineer especially when they are working in Web Development Domain. I will list down some of the key advantages of learning CSS:

Create Stunning Web site - CSS handles the look and feel part of a web page. Using CSS, you can control the colour of the text, the style of fonts, the spacing between paragraphs, how columns are sized and laid out, what background images or colours are used, layout designs, and variations in display for different devices and screen sizes as well as a variety of other effects.

Become a web designer - If you want to start a career as a professional web designer, HTML and CSS designing is a must skill.

Control web - CSS is easy to learn and understand but it provides powerful control over the presentation of an HTML document. Most commonly, CSS is combined with the markup languages HTML or XHTML.

Learn other languages - Once you understands the basic of HTML and CSS then other related technologies like JavaScript.

JavaScript:



JavaScript ("JS" for short) is a full-fledged dynamic programming language that, when applied to an HTML document, can provide dynamic interactivity on websites. It was invented by Brendan Eich, co-founder of the Mozilla project, the Mozilla Foundation, and the Mozilla Corporation.

JavaScript is incredibly versatile and beginner friendly. With more experience, you'll be able to create games, animated 2D and 3D graphics, comprehensive database-driven apps, and much more!

JavaScript itself is fairly compact yet very flexible. Developers have written a large variety of tools on top of the core JavaScript language, unlocking a vast amount of extra functionality with minimum effort. These include:

Browser Application Programming Interfaces (APIs) — APIs built into web browsers, providing functionality like dynamically creating HTML and setting CSS styles, collecting and manipulating a video stream from the user's webcam, or generating 3D graphics and audio samples.

T hird-party APIs — Allow developers to incorporate functionality in their sites from other content providers, such as Twitter or Facebook.

Third-party frameworks and libraries — you can apply these to your HTML to allow you to rapidly build up sites and applications.

Because this article is only supposed to be a light introduction to JavaScript, we are not going to confuse you at this stage by talking in detail about what the difference is between the core JavaScript language and the different tools listed above. You can learn all that in detail later on, in our JavaScript learning area, and in the rest of MDN.



Software Testing

**Software testing** is a process, to evaluate the functionality of a software application with an intent to find whether the developed software met the specified requirements or not and to identify the defects to ensure that the product is defect-free in order to produce the quality product.

# Definition:

**Software Testing Definition** according to **ANSI/IEEE 1059**standard – A process of analyzing a software item to detect the differences between existing and required conditions (i.e., defects) and to evaluate the features of the software item.



# Software Testing Types:

* Manual Testing: Manual testing is the process of testing software by hand to learn more about it, to find what is and isn’t working. This usually includes verifying all the features specified in requirements documents, but often also includes the testers trying the software with the perspective of their end user’s in mind. Manual test plans vary from fully scripted test cases, giving testers detailed steps and expected results, through to high-level guides that steer exploratory testing sessions. There are lots of sophisticated tools on the market to help with manual testing, but if you want a simple and flexible place to start, take a look at Test pad.
* Automation Testing: Automation testing is the process of testing the software using an automation tool to find the defects. In this process, testers execute the test scripts and generate the test results automatically by using automation tools. Some of the famous automation testing tools for functional testing are QTP/UFT and Selenium.

# Testing Methods:

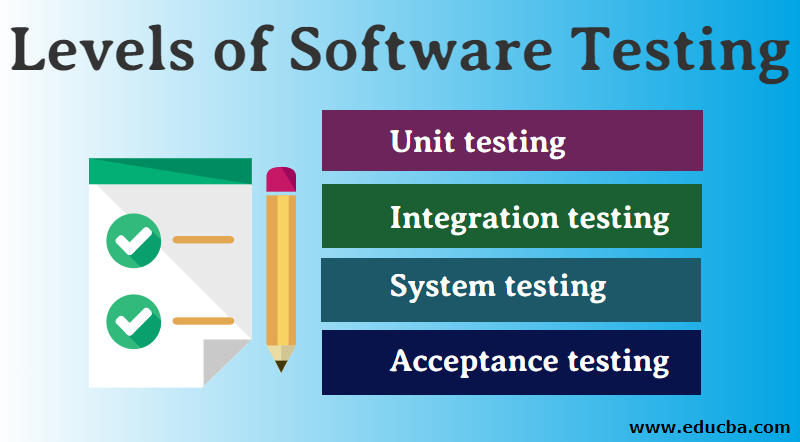
* Static Testing: It is also known as Verification in Software Testing. Verification is a static method of checking documents and files. Verification is the process, to ensure that whether we are building the product right i.e., to verify the requirements which we have and to verify whether we are developing the product accordingly or not.
* Dynamic Testing: It is also known as Validation in Software Testing. Validation is a dynamic process of testing the real product. Validation is the process, whether we are building the right product i.e., to validate the product which we have developed is right or not.

# Testing Approaches:

* White Box Testing: It is also called as Glass Box, Clear Box, Structural Testing. White Box Testing is based on application’s internal code structure. In white-box testing, an internal perspective of the system, as well as programming skills, are used to design test cases. This testing is usually done at the unit level.
* Black Box Testing: It is also called as Behavioral/Specification-Based/Input-Output Testing. Black Box Testing is a software testing method in which testers evaluate the functionality of the software under test without looking at the internal code structure.
* Grey Box Testing: Grey box is the combination of both White Box and Black Box Testing. The tester who works on this type of testing needs to have access to design documents. This helps to create better test cases in this process.

# Testing Level

* Unit Testing: Unit Testing is done to check whether the individual modules of the source code are working properly. i.e. testing each and every unit of the application separately by the developer in the developer’s environment. It is AKA Module Testing or Component Testing. To learn about Unit Testing, check out our detailed Unit Testing Guide
* Integration Testing: Integration Testing is the process of testing the connectivity or data transfer between a couple of unit tested modules. It is AKA I&T Testing or String Testing. It is subdivided into the Top-Down Approach, Bottom-Up Approach, and Sandwich Approach (Combination of Top-Down and Bottom-Up). To learn about Integration Testing, check out our detailed Integration Testing Guide
* System Testing (end to end testing): It’s a black box testing. Testing the fully integrated application this is also called as an end to end scenario testing. To ensure that the software works in all intended target systems. Verify thorough testing of every input in the application to check for desired outputs. Testing of the user’s experiences with the application.
* Acceptance Testing: To obtain customer sign-off so that software can be delivered and payments received. Types of Acceptance Testing are Alpha, Beta & Gamma Testing.



COST ESTIMATION AND MAINTAINANCE



# COST ESTIMATIONS:

Cost estimation can be defined as the approximate judgments of the costs for project. Cost estimation is usually measured in terms of effort. The effort is the amount of time for one person to work for a certain period of time. COCOMO is one the most widely used software estimation models in the world. The Constructive Cost Model (COCOMO) is a procedural software cost estimation model . COCOMO is used to estimate size, effort and duration based on the cost of the software.

COCOMO predicts the effort and schedule for a software product development based on inputs relating to the size of the software and a number of cost drivers that affect productivity.

# COCOMO has three different models that reflect the complexities:

**Basic Model:** This model would be applied early in a projects development. It will provide a rough estimate early on that should be refined later on with one of the other models.

**Intermediate Model:** This model would be used after you have more detailed requirements for a project.

**Detailed Model:** When design of the project is complete you can apply this model to further refine your estimate.

Within each of these models there are also three different modes. The mode you choose will depend on your work environment, and the size and constraints of the project itself.

The modes are:

**Organic:** This mode is used for “relativity small software teams developing software in a highly familiar, in-house environment”.

**Embedded:** Operating within tight constraints where the product is strongly tied to a “complex of hardware, software, regulations and operational procedures.

**Semi-detached:** An intermediate stage somewhere in between organic and embedded. Projects are usually of moderate size of up to 300,000 lines of code.

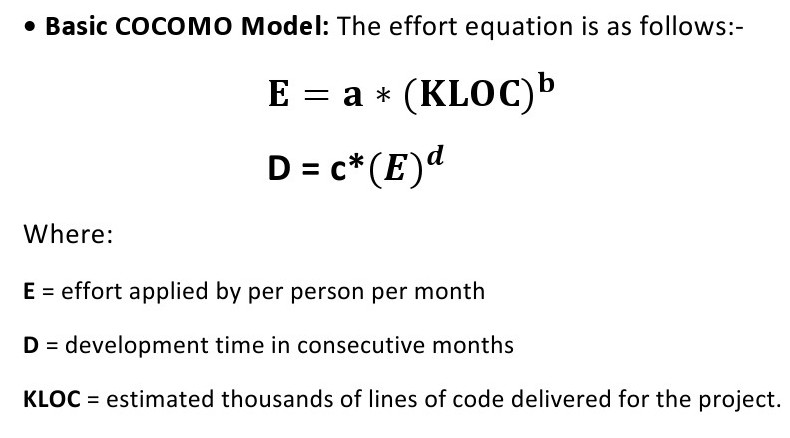
# TYPES OF COCOMO MODEL

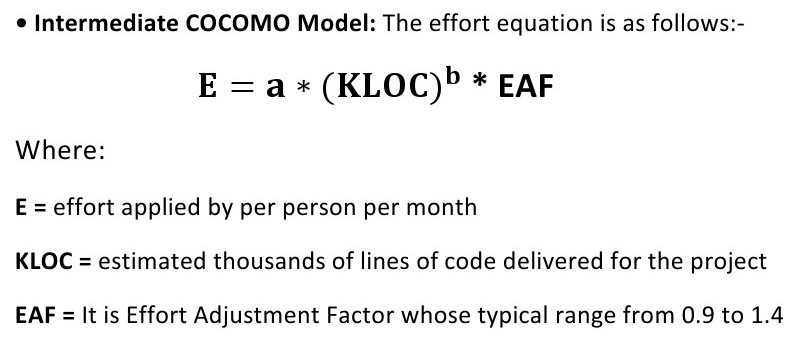
# D:\Study Material\SEM 4\SE\Types of COCOMO2.jpg

COCOMO

MODEL

* Basic Model: The basic COCOMO model estimates the software development effort using only Lines Of Code (LOC).

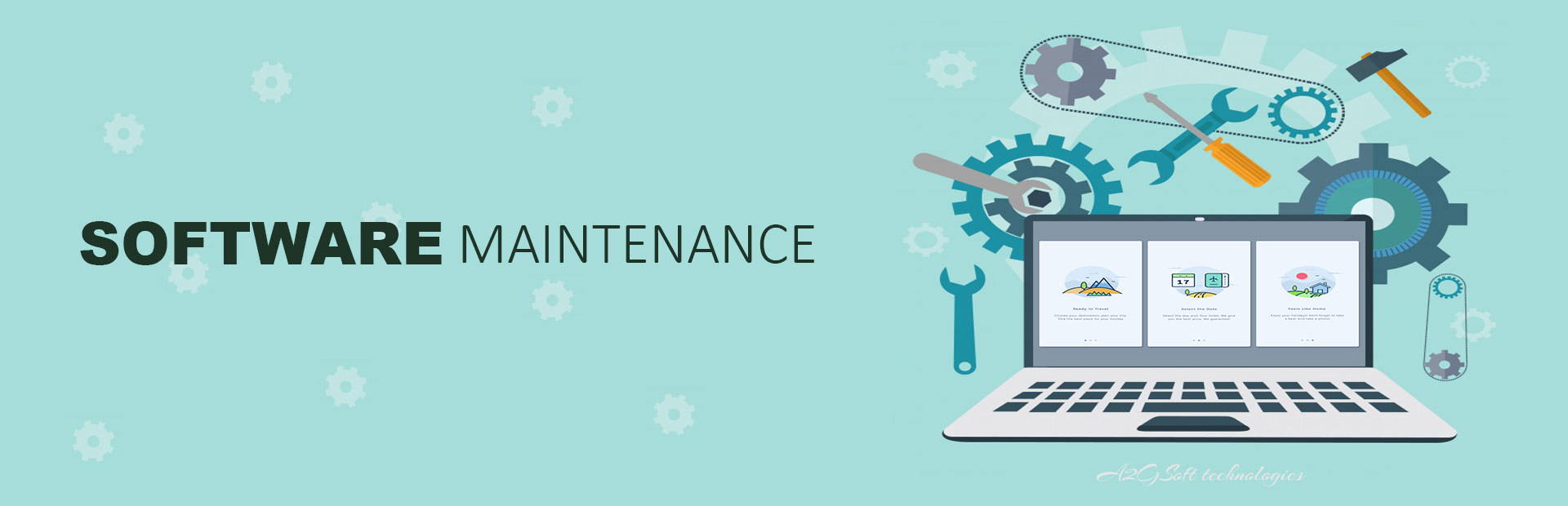


* Intermediate Model: This is an extension of basic COCOMO model. This estimation model makes use of set of cost driver attributes to compute the cost of software. The formula for effort calculation is: 
* Detailed Model: Detailed COCOMO incorporates all characteristics of the intermediate version with an assessment of the cost driver's impact on each step (analysis, design, etc.) of the software engineering process.

The detailed model uses different effort multipliers for each cost driver attribute. These Phase Sensitive effort multipliers are each to determine the amount of effort required to complete each phase. In detailed COCOMO, the whole software is divided into different modules and then we apply COCOMO in different modules to estimate effort and then sum the effort.

The effort is calculated as a function of program size and a set of cost drivers are given according to each phase of the software life cycle.

MAINTAINANCE



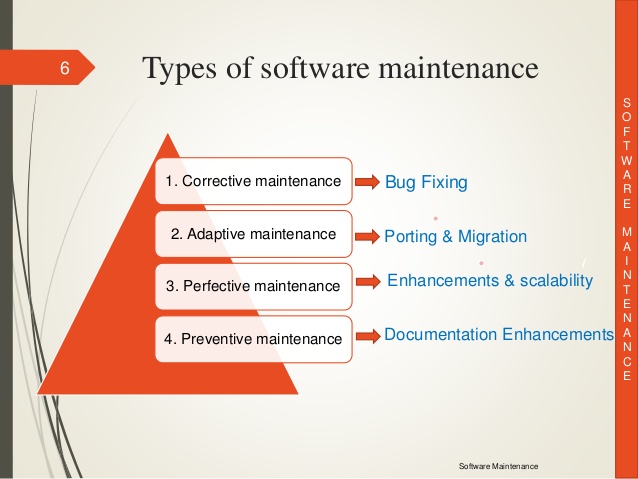
Software maintenance is widely accepted part of SDLC now a days. It stands for all the modifications and updates done after the delivery of software product. There are number of reasons, why modifications are required, some of them are briefly mentioned below:

* **Market Conditions**- Policies, which changes over the time, such as taxation and newly introduced constraints like, how to maintain bookkeeping, may trigger need for modification.
* **Client Requirements** - Over the time, customer may ask for new features or functions in the software.
* **Host Modifications**- If any of the hardware and/or platform (such as operating system) of the target host changes, software changes are needed to keep adaptability.
* **Organization Changes** - If there is any business level change at client end, such as reduction of organization strength, acquiring another company, organization venturing into new business, need to modify in the original software may arise.

# Types of maintenance

In a software lifetime, type of maintenance may vary based on its nature. It may be just a routine maintenance tasks as some bug discovered by some user or it may be a large event in itself based on maintenance size or nature. Following are some types of maintenance based on their characteristics:

* **Corrective Maintenance** - This includes modifications and updates done in order to correct or fix problems, which are either discovered by user or concluded by user error reports.
* **Adaptive Maintenance** - This includes modifications and updates applied to keep the software product up-to date and tuned to the ever changing world of technology and business environment.
* **Perfective Maintenance** - This includes modifications and updates done in order to keep the software usable over long period of time. It includes new features, new user requirements for refining the software and improve its reliability and performance.
* **Preventive Maintenance** - This includes modifications and updates to prevent future problems of the software. It aims to attend problems, which are not significant at this moment but may cause serious issues in future.



# Cost of Maintenance

Reports suggest that the cost of maintenance is high. A study on estimating software maintenance found that the cost of maintenance is as high as 67% of the cost of entire software process cycle.



On an average, the cost of software maintenance is more than 50% of all SDLC phases. There are various factors, which trigger maintenance cost go high, such as:

# **Real-world factors affecting Maintenance Cost**

* The standard age of any software is considered up to 10 to 15 years.
* Older software, which were meant to work on slow machines with less memory and storage capacity cannot keep themselves challenging against newly coming enhanced software on modern hardware.
* As technology advances, it becomes costly to maintain old software.
* Most maintenance engineers are newbie and use trial and error method to rectify problem.
* Often, changes made can easily hurt the original structure of the software, making it hard for any subsequent changes.
* Changes are often left undocumented which may cause more conflicts in future.

# Software-end factors affecting Maintenance Cost

* Structure of Software Program
* Programming Language
* Dependence on external environment
* Staff reliability and availability

# Maintenance Activities

IEEE provides a framework for sequential maintenance process activities. It can be used in iterative manner and can be extended so that customized items and processes can be included.



These activities go hand-in-hand with each of the following phase:

* **Identification & Tracing** - It involves activities pertaining to identification of requirement of modification or maintenance. It is generated by user or system may itself report via logs or error messages. Here, the maintenance type is classified also.
* **Analysis** - The modification is analyzed for its impact on the system including safety and security implications. If probable impact is severe, alternative solution is looked for. A set of required modifications is then materialized into requirement specifications. The cost of modification/maintenance is analyzed and estimation is concluded.
* **Design** - New modules, which need to be replaced or modified, are designed against requirement specifications set in the previous stage. Test cases are created for validation and verification.
* **Implementation** - The new modules are coded with the help of structured design created in the design step .Every programmer is expected to do unit testing in parallel.
* **System Testing** - Integration testing is done among newly created modules. Integration testing is also carried out between new modules and the system. Finally the system is tested as a whole, following regressive testing procedures.
* **Acceptance Testing** - After testing the system internally, it is tested for acceptance with the help of users. If at this state, user complaints some issues they are addressed or noted to address in next iteration.
* **Delivery** - After acceptance test, the system is deployed all over the organization either by small update package or fresh installation of the system. The final testing takes place at client end after the software is delivered.

Training facility is provided if required, in addition to the hard copy of user manual.

* **Maintenance management** - Configuration management is an essential part of system maintenance. It is aided with version control tools to control versions, semi-version or patch management.