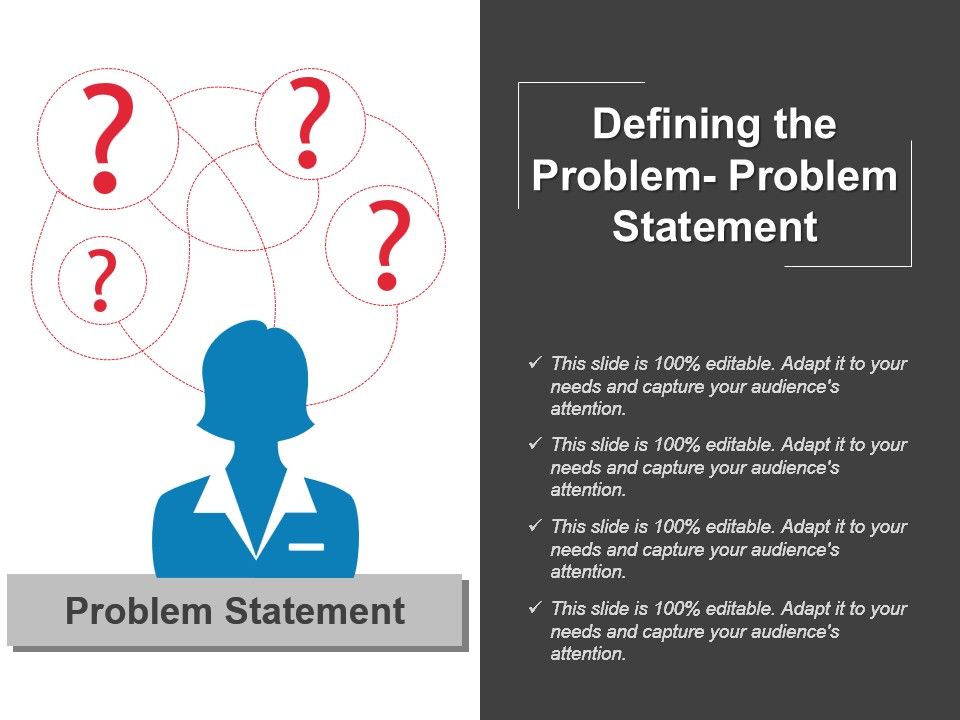
|  |  |  |
| --- | --- | --- |
| *SOFTWARE**ENGINEERING* | | |
|  | | | |
|  | ***Visual Mathematics***  ***FORMULA***   **DOCUMENTATION**  **PREPARED BY** 1. DEESHA RAWAT  2. GITANJALI PRASAD  3. GRACY PARKHE  **DATE - 25/04/2020**  SUBMITTED TO –  NIDHI NIGAM ***Index***  1. Problem Statement and solution 2. Layers of software engineering 3. Agile Manifesto 4. Comparative study of all models 5. Model: prototype Model 6. Functional Requirements 7. Non-Functional Requirements 8. UML Diagrams 9. Implementation 10. Our Prototype 11. Software testing 12. Cost Estimation 13. COCOMO model 14. Software Maintenance |

******

**VISUAL MATHMATICS FORMULA**

PROBLEM STATEMENT



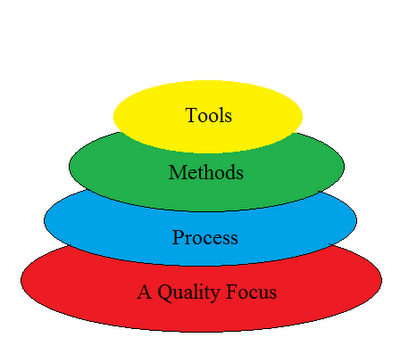
* Students have difficulties in understanding various math formula. It's easy to learn things but the better understood the more it is perceived and good to recall. Visualizing formula enables students to perceive things better and concepts are cleared more efficiently.

SOLUTION DOMAIN



* To understand things with greater efficiency and bring the concepts to practical problems is where, “VISUAL MATHEMATICS FORMULA”, comes into action. Visual mathematics formula shows the live presentation of various mathematical formula, through which the students can absorb the basic concepts and visualize their graphs.

**LAYERS OF SOFTWARE ENGINERRING**



**Tools**: This layer contains automated or semi-automated tools that offer support for the framework and the method each software engineering project will follow.

**Method**: This layer contains the methods, the technical knowledge and “how-to” in order to develop software.

**Process**: This layer consists of the framework that must be established for the effective delivery of software.

**A Quality Focus**: This layer is the fundamental layer for software engineering. As stated above it is of great importance to test the end product to see if it meets its specifications. Efficiency, usability, maintenance and reusability are some of the requirements that need to be met by new software.

The above four layers of software engineering for our software is described as follows:

**TOOLS:**

Our software includes frontend and backend which are developed using the following technologies. HTML, CSS, Java script.

**PROCESS**:

This layer consists of the framework that must be established for the effective delivery of software.

Visual mathematics formula uses HTML to create the web page background. The layout of the web page and layout of graph is made using html.

In software development, we are often interested in reducing repetition. We can achieve this through the use of Cascading Style Sheets. CSS specifically deals with the layout and customization of HTML elements. CSS is used to decorate the webpage from the selection fonts to their type, colour, size, the colour if the background, design effect is done using CSS. JavaScript is used to create graphs, and other visual effects on the website.

**A QUALITY FOCUS**:

This layer is the fundamental layer for software engineering. As stated above it is of great importance to test the end product to see if it meets its specifications. Efficiency, usability, maintenance and reusability are some of the requirements that need to be met by new software.

Having Tools, Methods and Processes laid out from the beginning of any software engineering process makes it an easier task for both developers and project managers to check the quality of the end product and deliver a more complex software on time by staying on budget.

Agile Manifesto

***What is Agile?***

Agile is the ability to create and respond to change. It is a way of dealing with, and ultimately succeeding in, an uncertain and turbulent environment.

The authors of the Agile Manifesto chose “Agile” as the label for this whole idea because that word represented the adaptiveness and response to change which was so important to their approach.

It’s really about thinking through how you can understand what’s going on in the environment that you’re in today, identify what uncertainty you’re facing, and figure out how you can adapt to that as you go along.

***What is Agile Software Development?***

Agile software development is more than frameworks such as Scrum, Extreme Programming or Feature-Driven Development (FDD).

Agile software development is more than practices such as pair programming, test-driven development, stand-ups, planning sessions and sprints.

Agile software development is an umbrella term for a set of frameworks and practices based on the values and principles expressed in the Manifesto for Agile Software Development and the 12 Principles behind it. When you approach software development in a particular manner, it’s generally good to live by these values and principles and use them to help figure out the right things to do given your particular context.

One thing that separates Agile from other approaches to software development is the focus on the people doing the work and how they work together. Solutions evolve through collaboration between self-organizing cross-functional teams utilizing the appropriate practices for their context.

There’s a big focus in the agile software development community on collaboration and the self-organizing team.

That doesn’t mean that there aren’t managers. It means that teams have the ability to figure out how they’re going to approach things on their own.

It means that those teams are cross-functional. Those teams don’t have to have specific roles involved so much as that when you get the team together, you make sure that you have all the right skill sets on the team.

There still is a place for managers. Managers make sure team members have, or obtain, the right skill sets. Managers provide the environment that allows the team to be successful. Managers mostly step back and let their team figure out how they are going to deliver products, but they step in when the teams try but are unable to resolve issues.

When most teams and organizations start doing agile software development, they focus on the practices that help with collaboration and organizing the work, which is great. However, another key set of practices that are not as frequently followed but should be are specific technical practices that directly deal with developing software in a way that help your team deal with uncertainty. Those technical practices are essential and something you shouldn’t overlook.

***Agile is a Mind-set***

Ultimately, Agile is a mind-set informed by the values contained in the Agile Manifesto and the 12 Principles behind the Agile Manifesto. Those values and principles provide guidance on how to create and respond to change and how to deal with uncertainty.

You could say that the first sentence of the Agile Manifesto encapsulates the whole idea: “We are uncovering better ways of developing software by doing it and helping others do it.”

When you face uncertainty, try something you think might work, get feedback, and adjust accordingly.

Keep the values and principles in mind when you do this. Let your context guide which frameworks, practices, and techniques you use to collaborate with your team and deliver value to your customers.



***The Authors***

1. Kent Beck
2. Mike Beedle
3. Arie van Bennekum
4. Alistair Cockburn
5. Ward Cunningham
6. Martin Fowler
7. Robert C. Martin
8. Steve Mellor
9. Dave Thomas
10. James Grenning
11. Jim Highsmith
12. Andrew Hunt
13. Ron Jeffries
14. Jon Kern
15. Brian Marick
16. Ken Schwaber
17. Jeff Sutherland

**Principles behind the Agile Manifesto**

*We follow these principles:*

*1.*

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

*2.*

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

*3.*

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

*4.*

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

*5.*

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

*6.*

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

*7.*

Working software is the primary measure of progress.

*8.*

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

*9.*

Continuous attention to technical excellence and good design enhances agility.

*10.*

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

*11.*

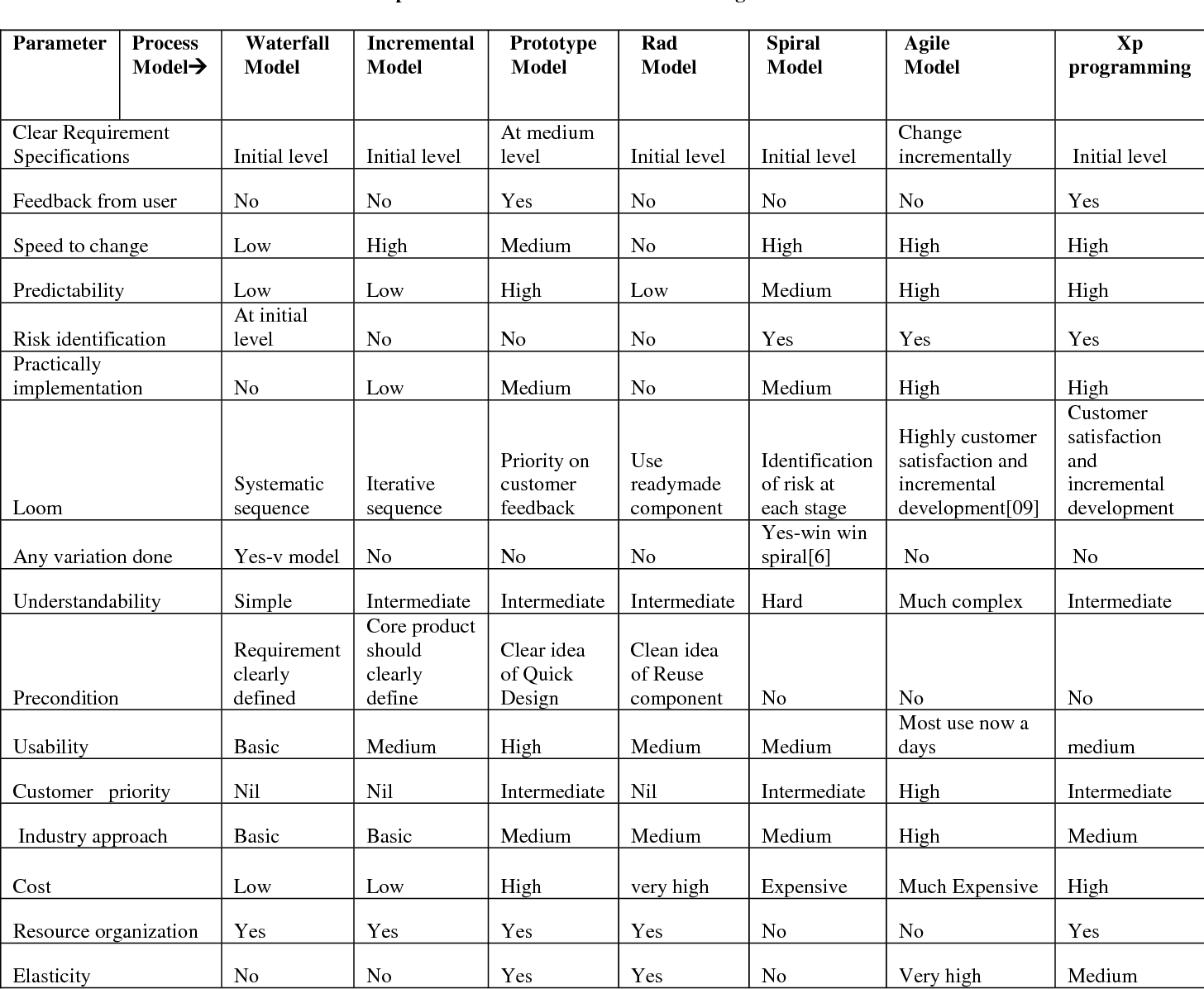
The best architectures, requirements, and designs emerge from self-organizing teams.

*12.*

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

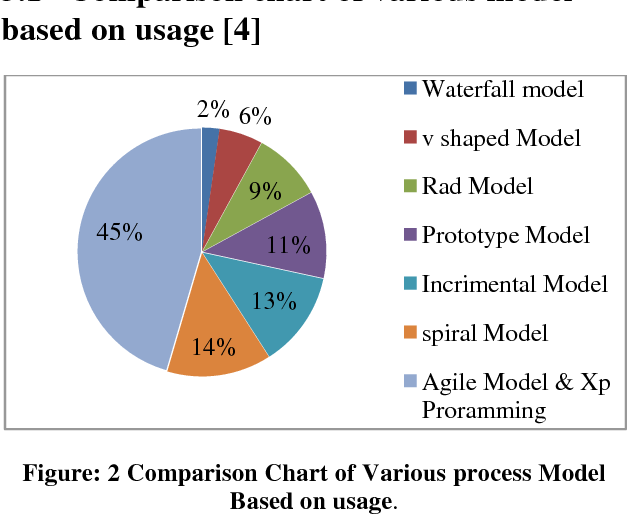
******

***Comparative Study between Software Models***



**

**

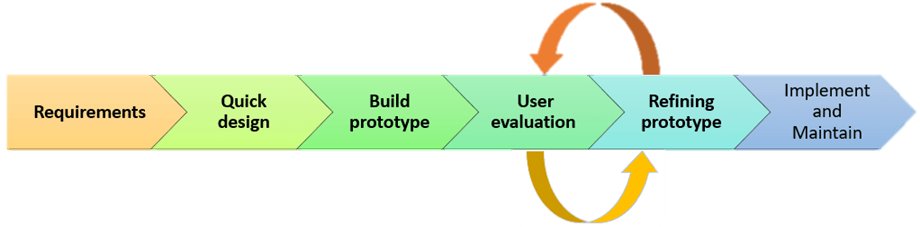


*MODEL:*

Prototype Model

Prototype methodology is defined as a Software Development model in which a prototype is built, test, and then reworked when needed until an acceptable prototype is achieved. It also creates a base to produce the final system.

Software prototyping model works best in scenarios where the project's requirement are not known. It is an iterative, trial, and error method which take place between the developer and the client.



Prototyping Model has following six SDLC phases as follow:

**Step 1: Requirements gathering and analysis**

A prototyping model starts with requirement analysis. In this phase, the requirements of the system are defined in detail. During the process, the users of the system are interviewed to know what is their expectation from the system.

**Step 2: Quick design**

The second phase is a preliminary design or a quick design. In this stage, a simple design of the system is created. However, it is not a complete design. It gives a brief idea of the system to the user. The quick design helps in developing the prototype.

**Step 3: Build a Prototype**

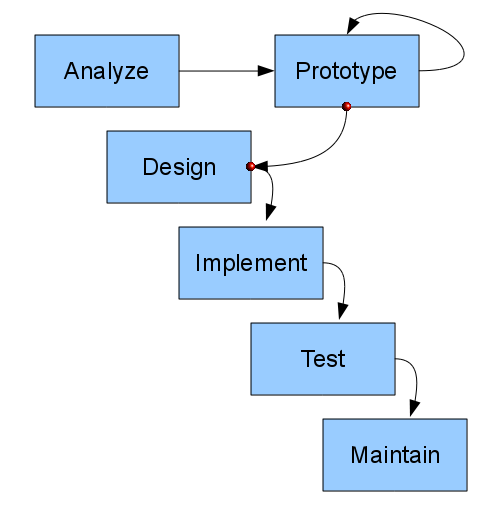
In this phase, an actual prototype is designed based on the information gathered from quick design. It is a small working model of the required system.

**Step 4: Initial user evaluation**

In this stage, the proposed system is presented to the client for an initial evaluation. It helps to find out the strength and weakness of the working model. Comment and suggestion are collected from the customer and provided to the developer.

**Step 5: Refining prototype**

If the user is not happy with the current prototype, you need to refine the prototype according to the user's feedback and suggestions.

This phase will not over until all the requirements specified by the user are met. Once the user is satisfied with the developed prototype, a final system is developed based on the approved final prototype.

**Types of Prototyping Models**

Four types of Prototyping models are:

1. Rapid Throwaway prototypes
2. Evolutionary prototype
3. Incremental prototype
4. Extreme prototype

* **Rapid Throwaway Prototype**

Rapid throwaway is based on the preliminary requirement. It is quickly developed to show how the requirement will look visually. The customer's feedback helps drives changes to the requirement, and the prototype is again created until the requirement is baselined.

In this method, a developed prototype will be discarded and will not be a part of the ultimately accepted prototype. This technique is useful for exploring ideas and getting instant feedback for customer requirements.

* **Evolutionary Prototyping**

Here, the prototype developed is incrementally refined based on customer's feedback until it is finally accepted. It helps you to save time as well as effort. That's because developing a prototype from scratch for every interaction of the process can sometimes be very frustrating.

This model is helpful for a project which uses a new technology that is not well understood. It is also used for a complex project where every functionality must be checked once. It is helpful when the requirement is not stable or not understood clearly at the initial stage.

* **Incremental Prototyping**

In incremental Prototyping, the final product is decimated into different small prototypes and developed individually. Eventually, the different prototypes are merged into a single product. This method is ****helpful to reduce the feedback time between the user and the application development team.

* **Extreme Prototyping:**

Extreme prototyping method is mostly used for web development. It is consists of three sequential phases.

Basic prototype with all the existing page is present in the HTML format.

You can simulate data process using a prototype services layer.

The services are implemented and integrated into the final prototype.

**Best practices of Prototyping**

Here, are a few things which you should watch for during the prototyping process:

You should use Prototyping when the requirements are unclear

It is important to perform planned and controlled Prototyping.

Regular meetings are vital to keep the project on time and avoid costly delays.

The users and the designers should be aware of the prototyping issues and pitfalls.

At a very early stage, you need to approve a prototype and only then allow the team to move to the next step.

In software prototyping method, you should never be afraid to change earlier decisions if new ideas need to be deployed.

You should select the appropriate step size for each version.

Implement important features early on so that if you run out of the time, you still have a worthwhile system

**Advantages of the Prototyping Model**

* 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 of the Prototyping Model**

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.

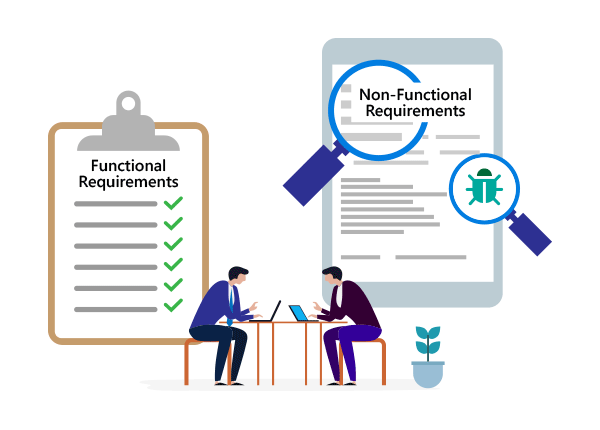
The client may lose interest in the final product when he or she is not happy with the initial prototype.

Developers who want to build prototypes quickly may end up building sub-standard development solutions.

*Why WE chose prototype model for our project…*

The requirenments for specification in prototype model are hat a medium level. Since, we take feedback from the use by presenting the prototype, we can learn about user requirenments, their expectations and also satisfy them by the progress of the job. Protoype makes the software predictable and the changes can be made faster.The elasticity is a better advange alg with the resource organisation, which makes it highly usable and adaptable.

* **WHAT IS FUNCTIONAL REQUIREMENT?**



In software engineering, a functional requirement defines a system or its component. It describes the functions a software must perform. A function is nothing but inputs, its behaviour, and outputs. It can be a calculation, data manipulation, business process, user interaction, or any other specific functionality which defines what function a system is likely to perform.  
Functional software requirements help you to capture the intended behaviour of the system. This behaviour may be expressed as functions, services or tasks or which system is required to perform.



WHAT IS NON-FUNCTIONAL REQUIREMENT?



A non-functional requirement defines the quality attribute of a software system. They represent a set of standards used to judge the specific operation of a system. Example, how fast does the website load?  
A non-functional requirement is essential to ensure the usability and effectiveness of the entire software system. Failing to meet non-functional requirements can result in systems that fail to satisfy user needs.  
Non-functional requirements allows you to impose constraints or restrictions on the design of the system across the various agile backlogs. Example, the site should load in 3 seconds when the number of simultaneous users are > 10000. Description of non-functional requirements is just as critical as a functional requirement.



**Visual Mathematics Formula**

Functional Requirements  
 visual mathematics formula does not require you to login or sign in. The only requirement you have to fulfill is provide a formula for the representation of the graph. Next to the formula you move ahead to the assignment of values so as to plot the graph. Plotting values can ensure the feasibility of the formula and preciseness of the formula. This way we can acknowledge the working of the formula and how else we can modify the formula.

Non-Functional Requirements   
 The graph can move as we desire. We can zoom the graph to study how precise it is. Published formulas can be tested and studied by plotting the using value substitution.

**VARIOUS SOFTWARE UML DIAGRAMS**

**Use case diagram –**

A use case diagram is a dynamic or behaviour diagram in [UML](https://www.smartdraw.com/uml-diagram/). Use case diagrams model the functionality of a system using actors and use cases. Use cases are a set of actions, services, and functions that the system needs to perform. In this context, a "system" is something being developed or operated, such as a web site. The "actors" are people or entities operating under defined roles within the system

## **Why Make Use Case Diagrams?**

Use case diagrams are valuable for visualizing the functional requirements of a system that will translate into design choices and development priorities.

They also help identify any internal or external factors that may influence the system and should be taken into consideration.

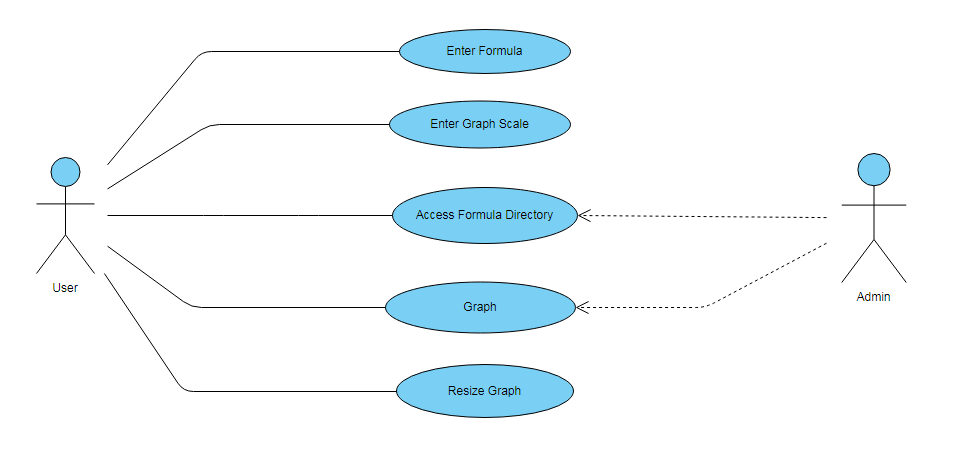
They provide a good high level analysis from outside the system. Use case diagrams specify how the system interacts with actors without worrying about the details of how that functionality is implemented.



We notice three main components of this uml diagram:

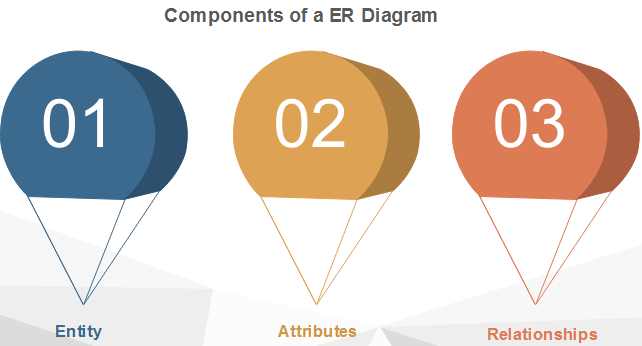
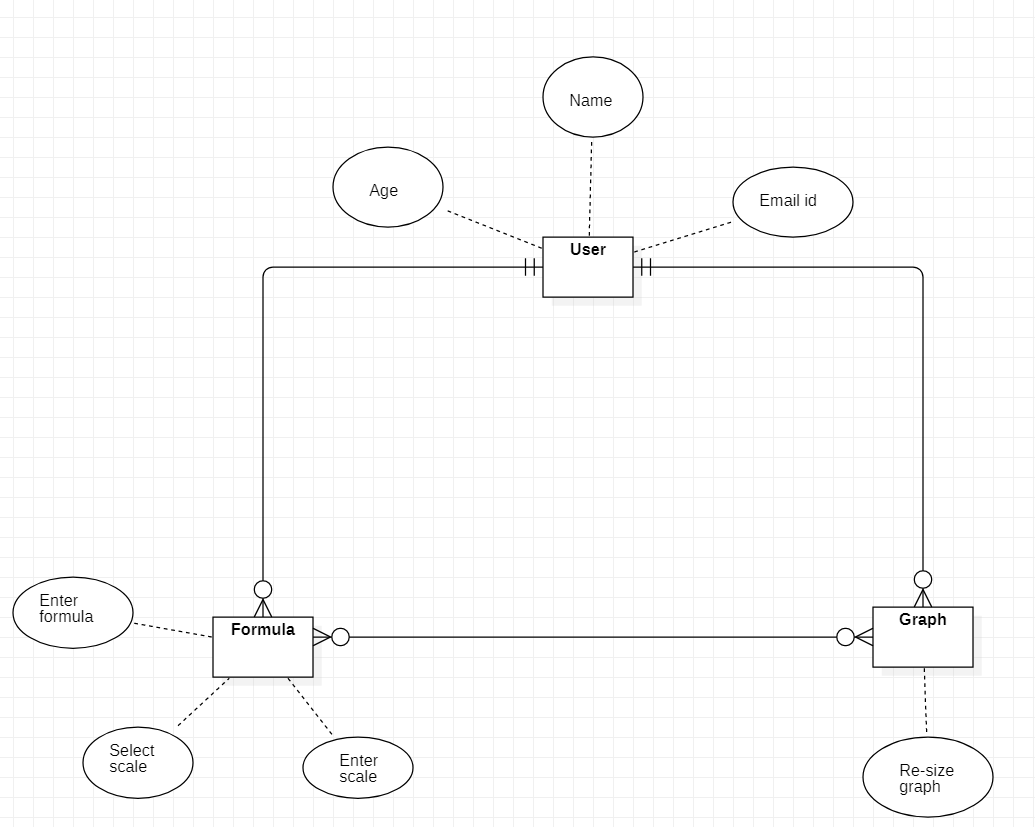
**Functional requirements** – Represented as use cases; a verb describing an action.  
**Actors** – They interact with the system; an actor can be a human being, an organization or an internal or external application.  
**Relationships between actors and use cases** – Represented using straight arrows.

**EXAMPLE OF USE CASE DIAGRAM**



**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 as they are commonly known.

**Components of a ER diagram**

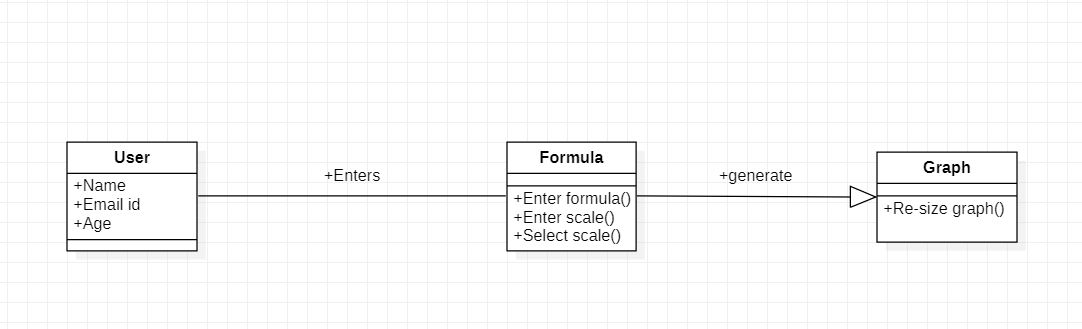


**CLASS DIAGRAM:** Class UML diagram is the most common diagram type for software documentation. Since most software being created nowadays is still based on the objected-oriented programming paradigm, using class diagrams to document the software turns out to be a common-sense solution.

**What is the purpose of class diagram?**

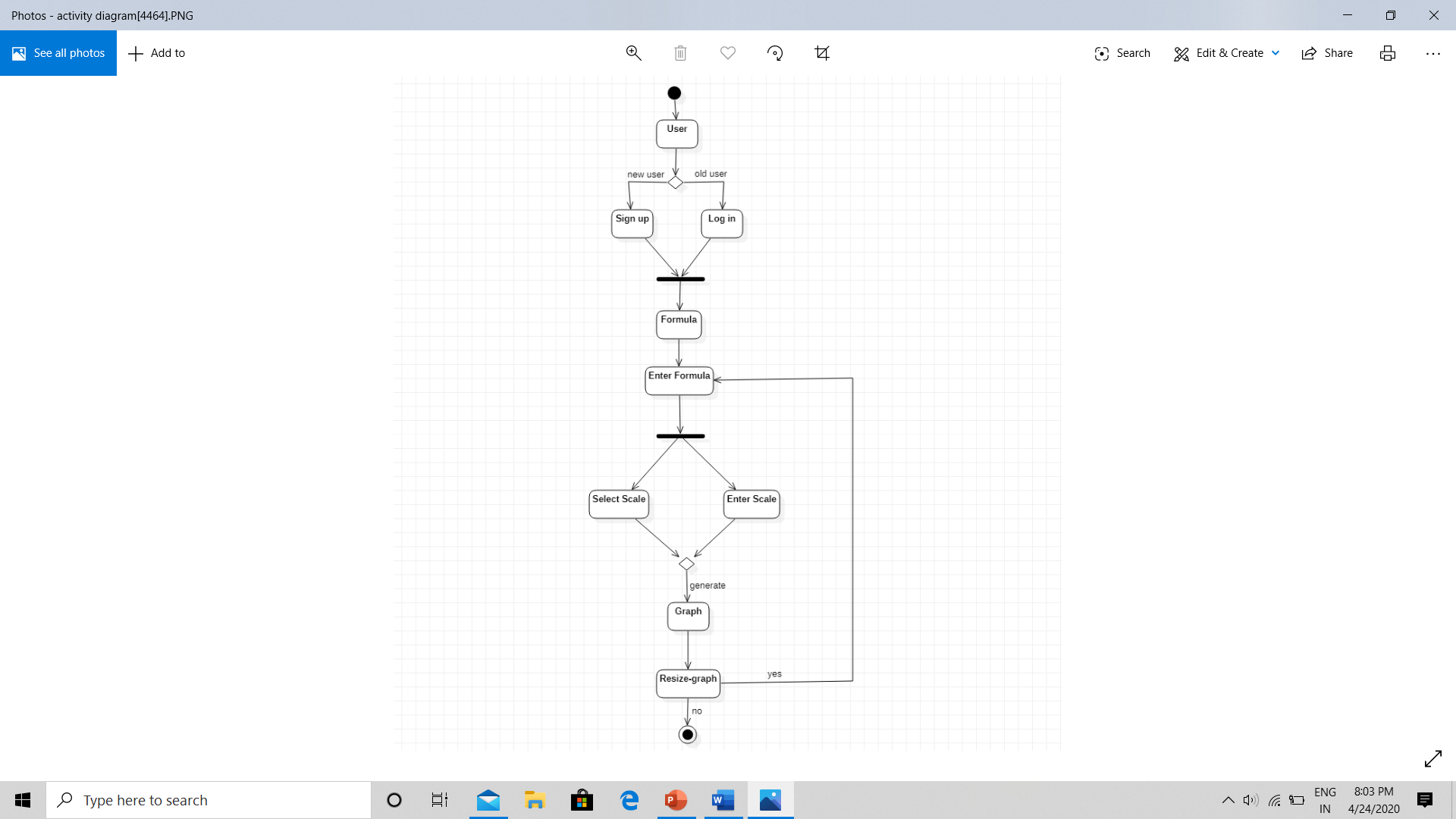


The purpose of the class diagram can be summarized as

* Analysis and design of the static view of an application.

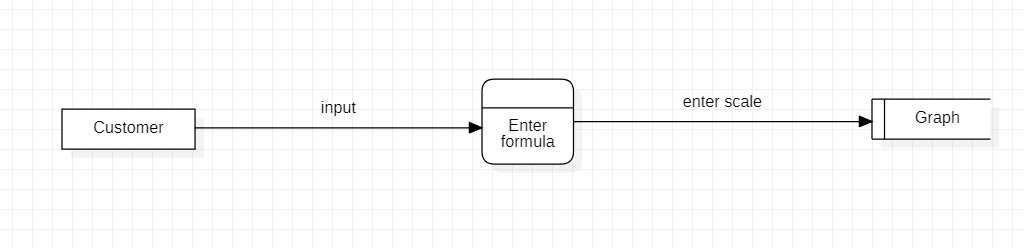
**ACTIVITY DIAGRAM:**

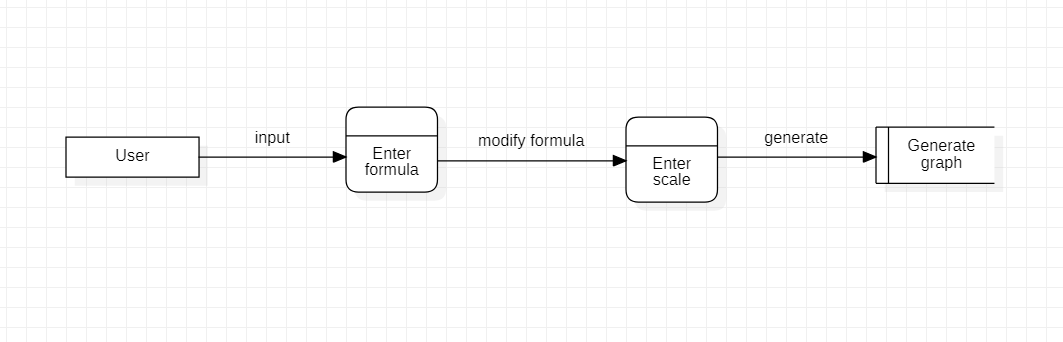
Activity diagrams are probably the most important UML diagrams for doing business process modelling. In software development, it is generally used to describe the flow of different activities and actions. These can be both sequential and in parallel. They describe the objects used, consumed or produced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



**Data Flow Diagram**

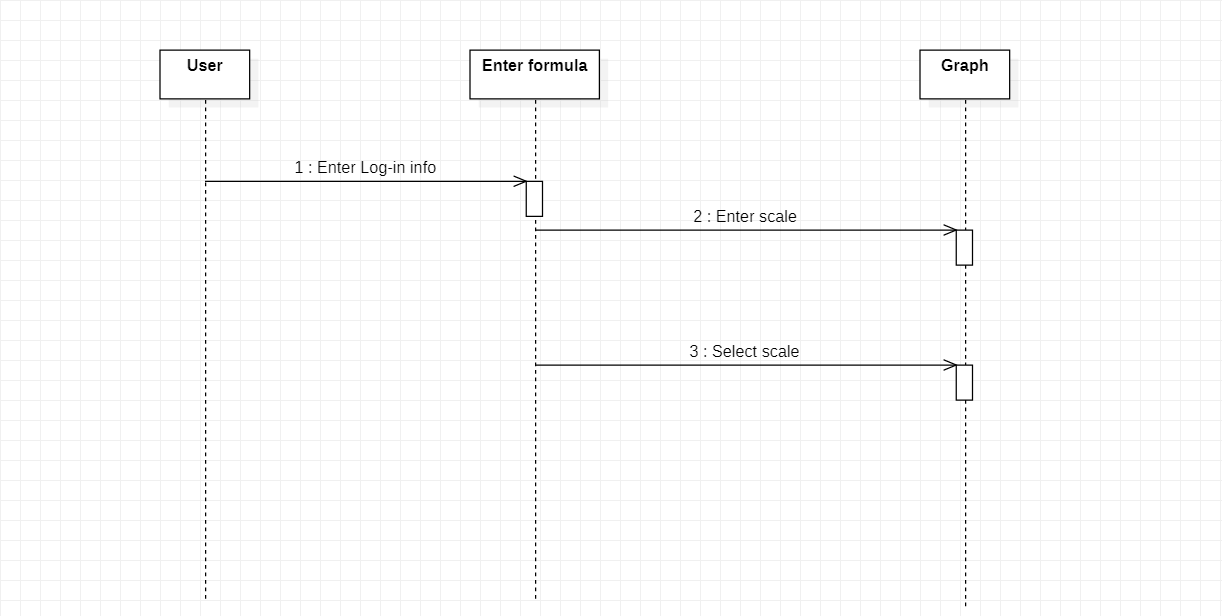
A data flow diagram is a way of representating a flow of a data of a process or a system. The DFD also provides information about the outputs and inputs of each entity and the process itself.

LEVEL 0 –

LEVEL 1-

**SEQUENCE DIAGRAM**

As the name suggests, sequence diagram describes the sequence 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.



**Implementation**

**SOFTWARE 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.

Third-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.

1. Semantic Web Page Layout with HTML

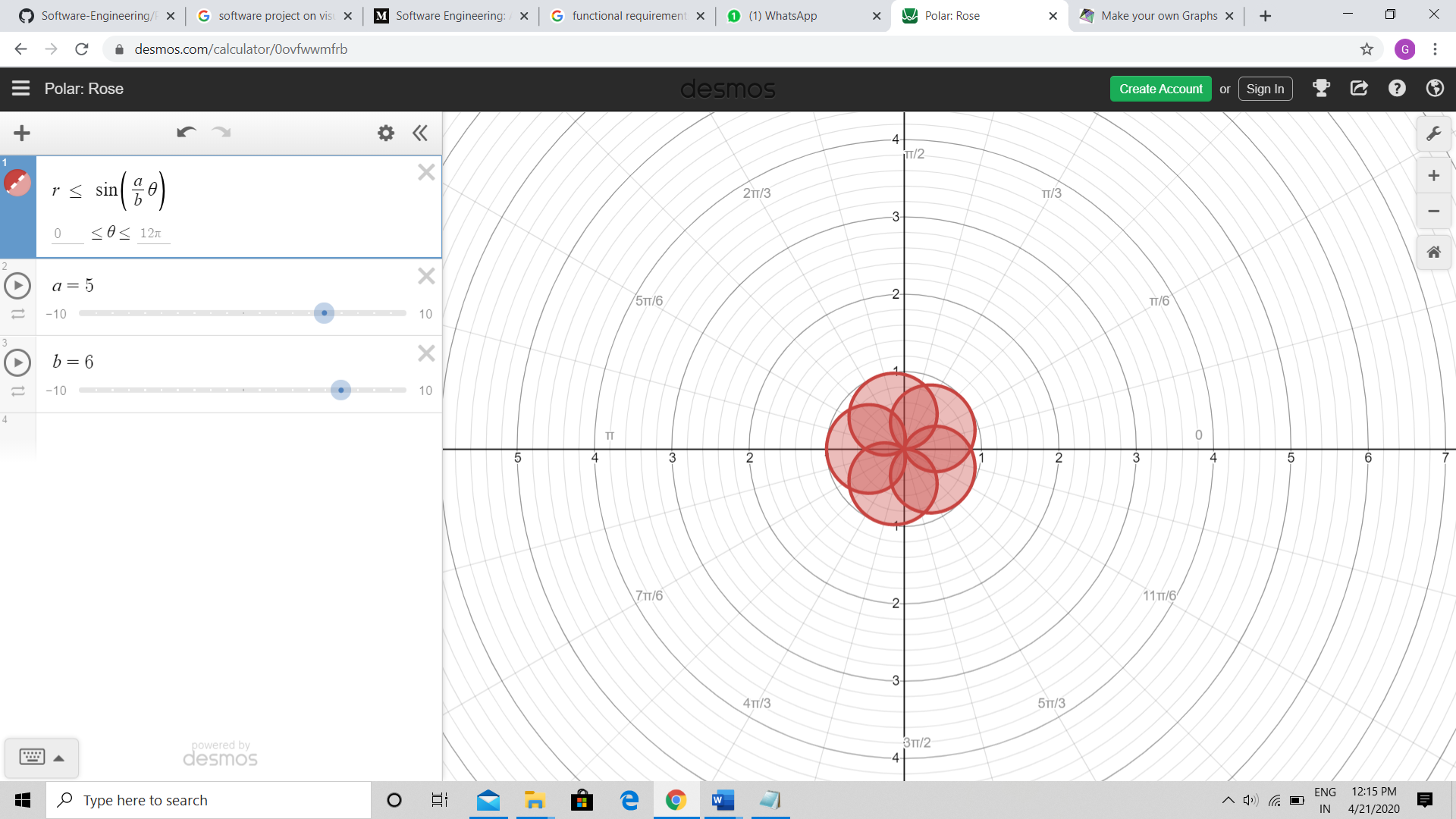
2. Stylizing HTML Content using CSS

3. Introduction to JavaScript, Basic Data Types + Operators

4. JavaScript Program Flow + Data Structures

5. Building Interactivity with DOM Manipulation

***Our Prototype:***



******

***Software Testing***

***What is 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.

******

***What is the purpose of software testing*?**

A primary purpose of testing is to detect software failures so that defects may be discovered and corrected. Testing cannot establish that a product functions properly under all conditions, but only that it does not function properly under specific condition.

## ***Why is Software Testing Important***?

Some of the reasons why software testing becomes very significant and integral part in the field of information technology are as follows.

1. Cost effectiveness
2. Customer Satisfaction
3. Security
4. Product Quality

***LEVELS OF SOFTWARE TESTING***:

****

***What are the levels of testing*?**

Tests are grouped together based on where they are added in SDLC or the by the level of detailing they contain. In general, there are four levels of testing: unit testing, integration testing, system testing, and acceptance testing. The purpose of Levels of testing is to make software testing systematic and easily identify all possible test cases at a particular level.

There are many different testing levels which help to check behavior and performance for software testing.

**HOW TO WRITE A TEST PLAN**

***Follow the seven steps below to create a test plan***:

****

**Testing Types For Visual Mathematics Formula**

***Functionality Testing*** - It is the act of testing the base functionality of the visual graph representation. It is to verify whether the software behaves in the same way as it is supposed to be like handling input formulas showing required output.

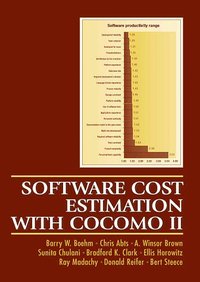
***Performance Testing*** - This test proves how efficient the software is. It tests the effectiveness and average time taken by the software to do desired task. It identifies performance matries like frequently entered formulas and stores the graph generated.

***Security & Portability*** - We need to perform security pass to check the authenticity of the user. It verifies whether the software is capable to run on multiple systems.

***Customer Satisfaction*** : It is verified with the number of users overtime.



***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 Model:***

The COCOMO (Constructive Cost Model) is one of the most popularly used software cost estimation models i.e. it estimates or predicts the effort required for the project, total project cost and scheduled time for the project. This model depends on the number of lines of code for software product development. It was developed by a software engineer Barry Boehm in 1981.

**What is COCOMO Model?**

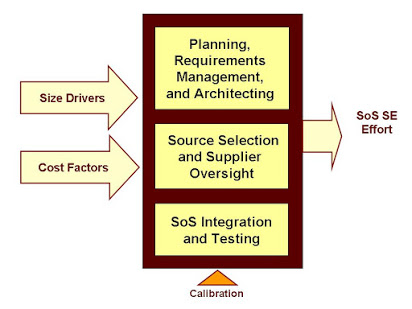
The COCOMO estimates the cost for software product development in terms of effort (resources required to complete the project work) and schedule (time required to complete the project work) based on the size of the software product. It estimates the required number of Man-Months (MM) for the full development of software products. According to COCOMO, there are three modes of software development projects that depend on complexity. Such as:

Organic Project: It belongs to small & simple software projects which are handled by a small team with good domain knowledge and few rigid requirements. Example: Small data processing or Inventory management system.

Semidetached Project: It is an intermediate (in terms of size and complexity) project, where the team having mixed experience (both experience & inexperience resources) to deals with rigid/non-rigid requirements. Example: Database design or OS development.

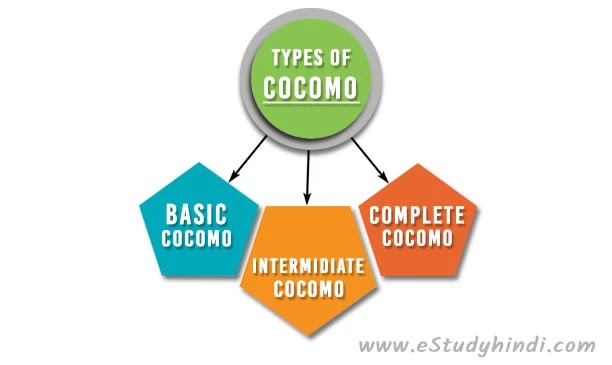
Embedded Project: This project having a high level of complexity with a large team size by considering all sets of parameters (software, hardware and operational). Example: Banking software or Traffic light control software.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **PROJECT TYPE** | **a** | **b** | **c** | **d** |
| **Organic** | 2.4 | 1.05 | 2.5 | 0.38 |
| **Semidetached** | 3 | 1.12 | 2.5 | 0.35 |
| **Embedded** | 3.6 | 1.2 | 2.5 | 0.32 |

****

****

***Types of COCOMO Model*:**

****

Depending upon the complexity of the project the COCOMO has three types. Such as:

**1. The Basic COCOMO**

It is the one type of static model to estimates software development effort quickly and roughly. It mainly deals with the number of lines of code and the level of estimation accuracy is less as we don’t consider the all parameters belongs to the project. The estimated effort and scheduled time for the project are given by the relation:

Effort (E) = a\*(KLOC) bMM  
Scheduled Time (D) = c\*(E) dMonths (M)

Where,

**E =** Total effort required for the project in Man-Months (MM).

**D =** Total time required for project development in Months (M).

**KLOC =** the size of the code for the project in Kilo lines of code.

**A, b, c, d** **=** the constant parameters for a software project.

**For Organic**

Effort (E) = a\*(KLOC) b= 2.4\*(300)1.05 = 957.61 MM

Scheduled Time (D) = c\*(E) d**=** 2.5\*(957.61)0.38 = 33.95 Months (M)

Avg. Resource Size = E/D = 957.61/33.95 = 28.21 Mans

Productivity of Software = KLOC/E = 300/957.61 = 0.3132 KLOC/MM = 313 LOC/MM

**For Semidetached**

Effort (E) = a\*(KLOC) b= 3.0\*(300)1.12 = 1784.42 MM

Scheduled Time (D) = c\*(E) d**=** 2.5\*(1784.42)0.35 = 34.35 Months (M)

**For Embedded**

Effort (E) = a\*(KLOC) b= 3.6\*(300)1.2 = 3379.46 MM

Scheduled Time (D) = c\*(E) d**=** 2.5\*(3379.46)0.32 = 33.66 Months (M)

**2. The Intermediate COCOMO**

The intermediate model estimates software development effort in terms of size of the program and other related cost drivers parameters (product parameter, hardware parameter, resource parameter, and project parameter) of the project. The estimated effort and scheduled time are given by the relationship:

Effort (E) = a\*(KLOC) b\*EAF MM  
Scheduled Time (D) = c\*(E) dMonths (M)

Where,

**E =** Total effort required for the project in Man-Months (MM).

**D =** Total time required for project development in Months (M).

**KLOC =** the size of the code for the project in Kilo lines of code.

**A, b, c, d =** the constant parameters for the software project.

**EAF** **=** It is an Effort Adjustment Factor, which is calculated by multiplying the parameter values of different cost driver parameters. For ideal, the value is 1.

Classification of Cost Drivers and their attributes:

**Product attributes –**

Required software reliability extent

Size of the application database

The complexity of the product

**Hardware attributes –**

Run-time performance constraints

Memory constraints

The volatility of the virtual machine environment

Required turnabout time

**Personnel attributes –**

Analyst capability

Software engineering capability

Applications experience

Virtual machine experience

Programming language experience

**Project attributes –**

Use of software tools

Application of software engineering methods

Required development schedule

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **COST DRIVERS PARAMETERS** | **VERY LOW** | **LOW** | **NOMINAL** | **HIGH** | **VERY HIGH** |
| **Product Parameter** | | | | | |
| **Required Software** | 0.75 | 0.88 | 1 | 1.15 | 1.4 |
| **Size of Project Database** | NA | 0.94 | 1.08 | 1.16 |
| **Complexity of The Project** | 0.7 | 0.85 | 1.15 | 1.3 |
| **Hardware Parameter** | | | | | |
| **Performance Restriction** | NA | NA | 1 | 1.11 | 1.3 |
| **Memory Restriction** | NA | NA | 1.06 | 1.21 |
| **virtual Machine Environment** | NA | 0.87 | 1.15 | 1.3 |
| **Required Turnabout Time** | NA | 0.94 | 1.07 | 1.15 |
| **Personnel Parameter** | | | | | |
| **Analysis Capability** | 1.46 | 1.19 | 1 | 0.86 | 0.71 |
| **Application Experience** | 1.29 | 1.13 | 0.91 | 0.82 |
| **Software Engineer Capability** | 1.42 | 1.17 | 0.86 | 0.7 |
| **Virtual Machine Experience** | 1.21 | 1.1 | 0.9 | NA |
| **Programming  Experience** | 1.14 | 1.07 | 0.95 | NA |
| **Project Parameter** | | | | | |
| **Software Engineering Methods** | 1.24 | 1.1 | 1 | 0.91 | 0.82 |
| **Use of Software Tools** | 1.24 | 1.1 | 0.91 | 0.83 |
| **Development Time** | 1.23 | 1.08 | 1.04 | 1.1 |



**3. The Detailed COCOMO**

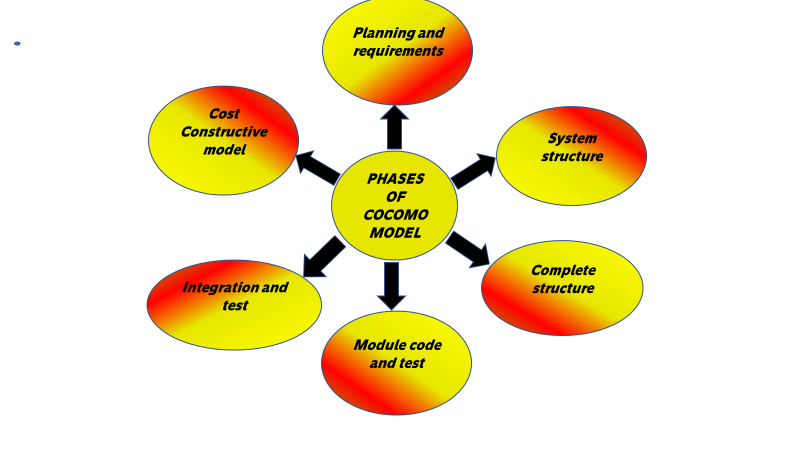
It is the advanced model that estimates the software development effort like Intermediate COCOMO in each stage of the software development life cycle process.

It incorporates all qualities of the standard version with an assessment of the cost driver’s effect on each method of the software engineering process. The detailed model uses various effort multipliers for each cost driver property. In detailed COCOCMO, the whole software is differentiated into multiple modules, and then we apply COCOMO in various modules to estimate effort and then sum the effort.

**The Six phases of detailed COCOMO are**:

1. Planning and requirements
2. System structure
3. Complete structure
4. Module code and test
5. Integration and test
6. Cost Constructive model

The effort is determined as a function of program estimate, and a set of cost drivers are given according to every phase of the software lifecycle.



***Software Maintenance***

Software maintenance is widely accepted part of SDLC now a days. It stands for all the modifications and updations 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.

**Need for Maintenance**

Software Maintenance is needed for:-

* Correct errors
* Change in user requirement with time
* Changing hardware/software requirements
* To improve system efficiency
* To optimize the code to run faster
* To modify the components
* To reduce any unwanted side effects.

Thus the maintenance is required to ensure that the system continues to satisfy user requirements.

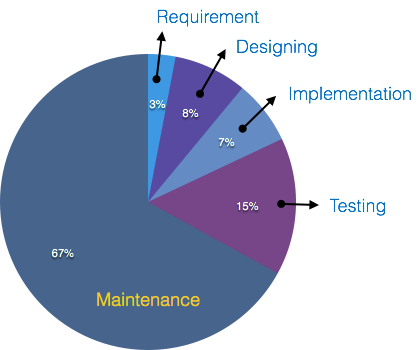
*****Types of maintenance***

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 updations 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 updations 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 updations 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**

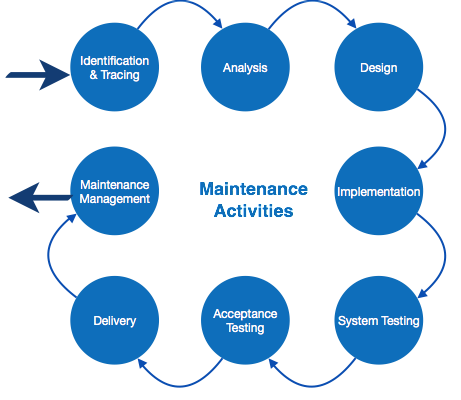
* The standard age of any software is considered up to 10 to 15 years.
* Older software’s, which were meant to work on slow machines with less memory and storage capacity cannot keep themselves challenging against newly coming enhanced softwares on modern hardware.
* As technology advances, it becomes 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 analysed 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 analysed 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.
* ***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.