

Course Title: Software Engineering

Course code: U-SOE-490

Definition of Software:

“Software is a collection of instructions and data that tell the computer how to work.”

“Software is a collection/set of instruction when executed it control the Hardware of computer.”

“Software is a set of instructions, data or programs used to operate computers and execute specific tasks.”

Software is Opposite of hardware, which describes the physical aspects of a computer, software is a generic term used to refer to applications, scripts and programs that run on a device.

Software can be thought of as the variable part of a computer, and hardware the invariable part.

Software is often divided into categories.

Application software refers to user-downloaded programs that fulfill a want or need. Examples of applications include office suites, database programs, web browsers, word processors, software development tools, image editors and communication platforms.

System software includes operating systems and any program that supports application software.

What is software engineering?

“Software Engineering is a systematic approach to develop a good quality software within the time and cost.”

Software engineering is an engineering branch associated with development of software product using well-defined scientific principles, methods and procedures. The outcome of software engineering is an efficient and reliable software product.

The meaning of software Engineering is by using well-defined

1. Scientific Principles.
2. Methods.
3. Procedures.

What will be the outcome of Software Engineering to develop a reliable and efficient software product?

Software characteristics:

As we know that software is any computer program which can also be defined as a set of instructions which are responsible for guiding the computer to perform certain tasks.

The following are the **characteristics of software**:

What are characteristics of software?

1. Software does not wear out
2. Software is not manufacture
3. Usability of Software
4. Reusability of components
5. Flexibility of software
6. Maintainability of software
7. Portability of software
8. Reliability of Software

1. Software does not wear out:

Different things like clothes, shoes, ornaments do wear out after some time. But, software once created never wears out. It can be used for as long as needed and in case of need for any updating, required changes can be made in the same software and then it can be used further with updated features.

2. Software is not manufactured:

Software is not manufactured but is developed. So, it does not require any raw material for its development.

3. Usability of Software:

The usability of the software is the simplicity of the software in terms of the user. The easier the software is to use for the user, the more is the usability of the software as more number of people will now be able to use it and also due to the ease will use it more willingly.

4. Reusability of components:

As the software never wears out, neither do its components, i.e. code segments. So, if any particular segment of code is required in some other software, we can reuse the existing code from the software in which it is already present. This reduced our work and also saves time and money.

5. Flexibility of software:

A software is flexible. What this means is that we can make necessary changes in our software in the future according to the need of that time and then can use the same software then also.

6. Maintainability of software:

Every software is maintainable. This means that if any errors or bugs appear in the software, then they can be fixed.

7. Portability of software:

Portability of the software means that we can transfer our software from one

platform to another that too with ease. Due to this, the sharing of the software among the developers and other members can be done flexibly.

8. Reliability of Software:

This is the ability of the software to provide the desired functionalities under every condition. This means that our software should work properly in each condition.

Software applications:

Explain software applications in detail.

What are the software applications?

Explain software applications with suitable examples.

Different Types of Software Application

- 1. System Software:** A collection of programs written to service other programs. Compiler, device driver, editors, file management. (Operating Systems)

Operating system is a program when it executes it control all the software as well as hardware of the computer.

e.g Windows 10, XP 2000, windows server operating system, MAC, Linux and so on.

- 2. Application software or standalone program:** It solves a specific Business needs. It is needed to convert the business function in real time. Example -point of sale, Transaction processing, real time manufacturing control. Word, Excel, PowerPoint

- 3. Scientific / Engineering Software:** Applications like based on astronomy, automotive stress analysis, molecular Biology, volcanology, space Shuttle orbital dynamic, automated manufacturing.

- 4. Embedded Software:** There are software control systems that control and manage hardware devices. Example- software in mobile phone, software in Anti-Lock Braking in car, software in microwave oven to control the cooking process.
- 5. Product Line Software:** It is designed to provide a specific capability for used by many different customers. It can focus on unlimited or esoteric Marketplace like inventory control products. Or address mass market place like: Spreadsheets, computer graphics, multimedia, entertainment, database management, personal, business financial applications.
- 6. Web application:** It is also called "web apps ", are evolving into sophisticated computing environment that not only provide standalone features, computing functions, and content to the end user but also are integrated with corporate database and business applications.
- 7. Artificial intelligence software:** This include- robotic, expert system, pattern recognition, image and voice, artificial neural network, game playing, theorem proving ... It solves Complex problems.

Definition of Software

Software is any computer program that enables you to perform certain specific tasks on your computer. It directs the function of all the peripheral devices of the computer, including your mouse, hard drive, monitor, and keyboard, etc. Without software, it would be impossible to operate the hardware. A software comprises two major categories: system software and application software.

System Software and Types of System Software

System software is the interface, or the intermediary. It lets you communicate between the other software and the hardware. There are five types of system software. These are intended to control and coordinate the functions and procedures of computer hardware and, thus, streamline the interaction between software, hardware, and the user.

Types of system software:

- **Operating system (OS):** It is a system software kernel that is installed first on the computer to allow applications and devices to be identified and become functional. Being the first layer of software, it is loaded into memory each time a computer is powered up. Some types of operating systems are real-time OS, single-user and single-task OS, network OS, and mobile OS.
- **Device drivers:** These bring computer devices and peripherals to life. Device drivers connect components and external add-ons so that they can perform their intended tasks. Without the driver, the operating system would not assign any duties. The mouse, keyboard, speakers, and printer are a few examples of devices that need drivers to function.
- **Firmware:** It is the operational software embedded within a ROM, flash, or EPROM memory chip for the operating system to identify it. It manages and controls all the activities of a single hardware directly. Firmware can be upgraded easily without swapping semiconductor chips.
- **Programming language translators:** These are intermediate programs used to translate high-level language source code to machine language code. Popular language translators include assemblers, compilers, and interpreters. They may be used to perform complete translation of program codes or they may translate each instruction one at a time.
- **Utilities:** It is intended for diagnostic and maintenance tasks for the computer. Their tasks may range from crucial data security to disk drive defragmentation.

Application Software and Types of Application Software

Application software is a computer program that performs a specific function, be it educational, personal, or business. It is also known as an end-user program or a productivity program. You can think of system software as a cake and the application software as the frosting on top of it. It is the application software (frosting) that you, the user, get to see upfront when you are working with that software.

Each of the computer application software programs is developed to assist you with a particular process that may be related to creativity, productivity, or better communication. It helps you in completing your tasks, be it jotting down notes, completing your online research, setting an alarm, keeping an account log, and even playing games. Unlike system software, computer application software programs are specific in their functionality and do the job that they are designed to do. For instance, a browser is an application used specifically for browsing the Internet. Similarly, MS PowerPoint is an application designed specifically for making presentations. Application software is also termed as a non-essential software. As we saw in the analogy of a cake and its frosting, its requirement is subjective and its absence does not affect the functioning of the system. All the apps that we see on our smartphones are examples of application software.

Picking up the right application for personal or business use can improve function and efficiency. If you do not learn about your options, you may end up with something that does not benefit your business, which will cost you time and resources and affect your productivity. Knowing what types of application software are available to you will help you make an informed decision.

The application software list includes:

- Word processors
- Graphics software
- Database software
- Spreadsheet software
- Presentation software
- Web browsers
- Enterprise software
- Information worker software
- Multimedia software
- Education and reference software
- Content access software

The categories of application software you are required to use depends on your needs, some of which are detailed below:

1. Presentation software:

Presentation software enables you to put forth your thoughts and ideas with ease and with good clarity by using visual information. It lets you display the information in the form of slides. You can make your slide more informative and more engrossing by adding text, images, graphs, and videos. It has three components:

1. Text editor to input and format text
2. Insert graphics, text, video, and multimedia files
3. Slideshow to display the information

2. Web browsers:

These software applications are used to browse the Internet enabling you to locate and retrieve data across the web. The most popular ones are Google Chrome and Internet Explorer.

3. Multimedia software:

This lets you create or record images, and create audio or video files. This software is extensively used in animation, graphics, image, and video editing. Popular examples are the VLC media player and Windows media player.

4. Education and reference software:

This application software, also termed as academic software, is specifically designed to facilitate the learning of a particular subject. Various kinds of tutorial software are included in this category. Some of these are JumpStart titles, MindPlay, and Kid Pix.

5. Graphics software:

Graphics software allows you to edit or make changes in visual data or images. It comprises illustration and picture editor software. Adobe Photoshop and PaintShop Pro are a few examples of graphics software.

6. Spreadsheet software:

Spreadsheet software is used to perform calculations. In this software, data is stored in a table format. The intersecting area, called cells, are separated to define fields such as text, date, time, and number. It allows users to provide formulas and functions to perform calculations. Microsoft Excel is one good example of spreadsheet software.

7. Database software:

Database software is used to create and manage a database. Also known as a DBMS (Database Management System), it helps you organize your data. So, when you run an application, data is fetched from the database, modified, and is stored back in the database. Oracle, MySQL, Microsoft SQL Server, PostgreSQL, MongoDB, and IBM Db2 are some popular databases.

8. Word processing software:

It is used to format and manipulate text, thus, creating memos, letters, faxes, and documents. Word processing software is also used to format and beautify the text. It provides you a whole lot of features aside from thesaurus and synonyms and antonyms. Along with Word Art features, the font option lets you change font color, effect, and style as per your choice. Grammar and spell-check options are also available to check for errors.

9. Simulation software:

Simulation software is used in the fields of engineering, education, testing, and video games, etc. It is used where work on the actual system is unacceptable, inaccessible, or maybe dangerous. It is a program that lets you study or observe an operation, or phenomenon through simulation without actually doing that operation. The best examples of the simulation are in the field of robotics, flight systems, and weather forecast, etc.

Apart from these, there are several others in the category of application software that serve specific purposes.

However, application software can also be classified based on their share ability and availability. Some such categories are:

10. Freeware:

As the very name indicates, it is available free of cost. You can download it from the Internet and use it without any fee. However, this software does not allow you to modify it or charge a fee for distributing it. Adobe Reader and Skype are good examples of this software.

11. Shareware:

This is distributed freely to the users on a trial basis, usually with a limited time offer. The users are expected to pay if they want to continue to use the software. Some examples of shareware are WinZip and Adobe Acrobat.

12. Open source:

This type of software is available along with the source code that allows you to modify the software, and even add features to the software. These could either be free or paid. Moodle and Apache Web Server are some examples.

13. Closed source:

Most of the software packages that you use belong to this category. These are usually chargeable and have intellectual property rights or patents over the source code. It usually comes with restricted use.

Functions of an Application Software

Application software programs are designed to facilitate a large number of functions. Some of these include managing information, constructing visuals, manipulating data, coordinating resources, and calculating figures.

Future of Software

The world is becoming increasingly computerized. With that growth, different types of application software will continue to evolve. Several software applications are being created as it is a great financial option for the users as well as the creators. The demand for custom software development, tailored to the requirements of a business is increasing now more than ever before. From simple customizations to full-cycle software development, Figment helps address all your specific business requirements, leveraging our software development capabilities.

If you have further questions or want to get started on choosing and implementing the right software for your business.

What is software engineering?

“Software Engineering is a systematic approach to develop a good quality software within the time and cost.”

Software engineering is an engineering branch associated with development of software product using well-defined scientific principles, methods and procedures. The outcome of software engineering is an efficient and reliable software product.

2. Software Process:

What is software Process?

Explain software process with example.

Why we use software process.

Software Process Models:

Software Processes is a **clear set of activities** for specifying, designing, implementing and testing software systems.

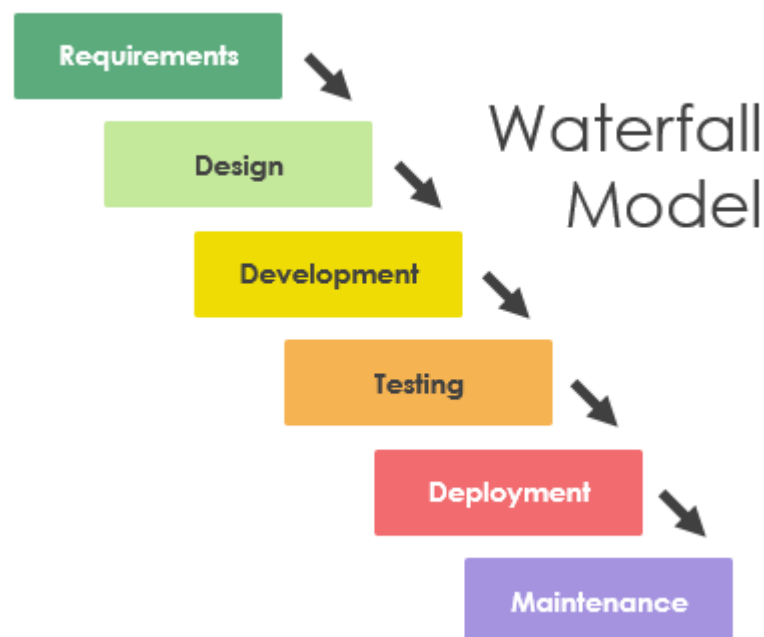
A software process model **is an abstract** representation of a process that presents a description of a process from some particular perspective. There are many different software processes but all involve:

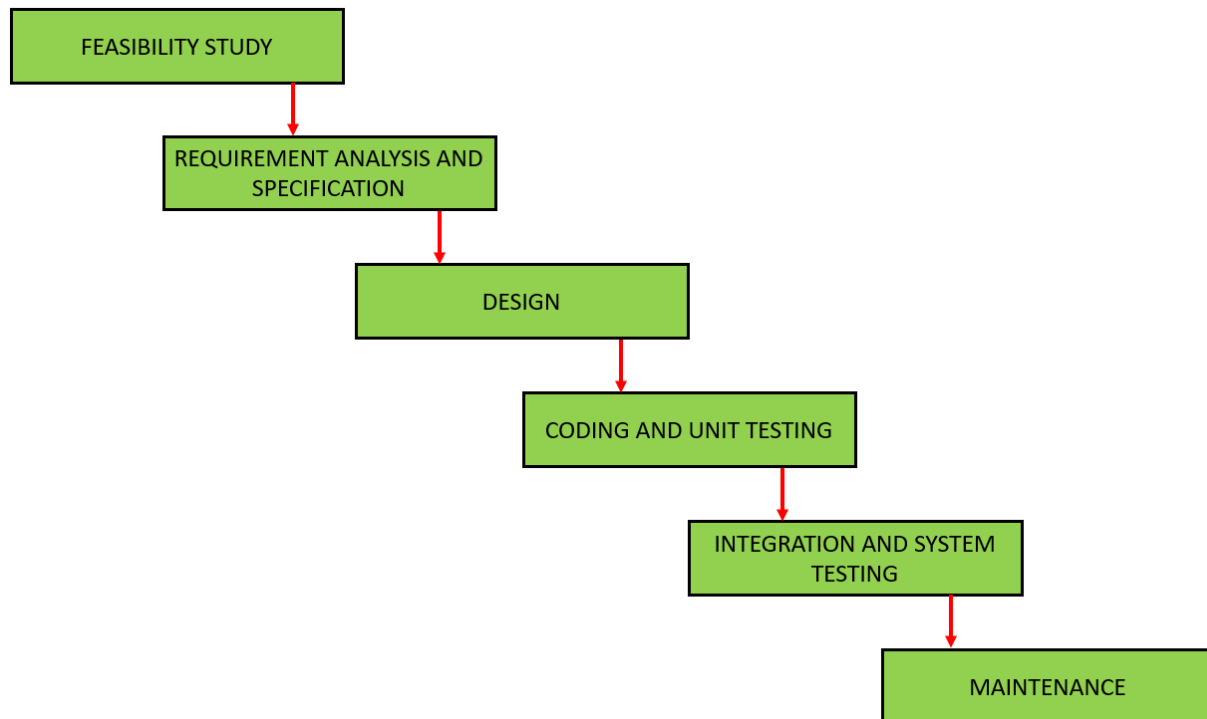
1. Specification – defining what the system should do;
2. Design and implementation – defining the organization of the system and implementing the system;
3. Validation – checking that it does what the customer wants;
4. Evolution – changing the system in response to changing customer needs.

Waterfall model :(First model)

The Waterfall model first introduced by Dr. Winston W. Royce in a paper published in 1970.

Classical waterfall model is the basic **software development life cycle** model. It is very simple but idealistic. Earlier this model was very popular but nowadays it is not used. But it is very important because all the other software development life cycle models are based on the classical waterfall model. Classical waterfall model divides the life cycle into a set of phases. This model considers that one phase can be started after completion of the previous phase. That is the output of one phase will be the input to the next phase. Thus the development process can be considered as a sequential flow in the waterfall. Here the phases do not overlap with each other. The different sequential phases of the classical waterfall model are shown in the below figure:





The waterfall model is a breakdown of project activities into linear sequential phases, where each phase depends on the deliverables of the previous one and corresponds to a specialization of tasks. The approach is typical for certain areas of engineering design.

Let us now learn about each of these phases in brief details:

1. **Feasibility Study:** The main goal of this phase is to determine whether it would be financially and technically feasible to develop the software. The feasibility study involves understanding the problem and then determine the various possible strategies to solve the problem. These different identified solutions are analyzed based on their benefits and drawbacks, The best solution is chosen and all the other phases are carried out as per this solution strategy.
2. **Requirements analysis and specification:** The aim of the requirement analysis and specification phase is to understand the exact requirements of the customer and document them properly. This phase consists of two different activities.
 - **Requirement gathering and analysis:** Firstly all the requirements regarding the software are gathered from the customer and then the gathered requirements are analyzed. The goal of the analysis part is to remove incompleteness (an incomplete requirement is one in which some parts of the actual requirements have been omitted) and inconsistencies

(inconsistent requirement is one in which some part of the requirement contradicts with some other part).

- **Requirement specification:** These analyzed requirements are documented in a software requirement specification (SRS) document. SRS document serves as a contract between development team and customers. Any future dispute between the customers and the developers can be settled by examining the SRS document.
- 3. **Design:** The aim of the design phase is to transform the requirements specified in the SRS document into a structure that is suitable for implementation in some programming language.
- 4. **Coding and Unit testing:** In coding phase software design is translated into source code using any suitable programming language. Thus each designed module is coded. The aim of the unit testing phase is to check whether each module is working properly or not.
- 5. **Integration and System testing:** Integration of different modules are undertaken soon after they have been coded and unit tested. Integration of various modules is carried out incrementally over a number of steps. During each integration step, previously planned modules are added to the partially integrated system and the resultant system is tested. Finally, after all the modules have been successfully integrated and tested, the full working system is obtained and system testing is carried out on this. System testing consists three different kinds of testing activities as described below:

- **Alpha testing:** Alpha testing is the system testing performed by the development team.
 - **Beta testing:** Beta testing is the system testing performed by a friendly set of customers.
 - **Acceptance testing:** After the software has been delivered, the customer performed the acceptance testing to determine whether to accept the delivered software or to reject it.
1. **Maintenance:** Maintenance is the most important phase of a software life cycle. The effort spent on maintenance is the 60% of the total effort spent to develop a full software. There are basically three types of maintenance :
 - **Corrective Maintenance:** This type of maintenance is carried out to correct errors that were not discovered during the product development phase.
 - **Perfective Maintenance:** This type of maintenance is carried out to enhance the functionalities of the system based on the customer's request.

- **Adaptive Maintenance:** Adaptive maintenance is usually required for porting the software to work in a new environment such as work on a new computer platform or with a new operating system.

Advantages of Classical Waterfall Model

Classical waterfall model is an idealistic model for software development. It is very simple, so it can be considered as the basis for other software development life cycle models. Below are some of the major advantages of this SDLC Software Development life cycle (SDLC) model:

- This model is very simple and is easy to understand.
- Phases in this model are processed one at a time.
- Each stage/phase/part/section in the model is clearly defined.
- This model has very clear and well understood milestones.
- Process, actions and results are very well documented.
- Reinforces good habits: define-before- design, design-before-code.
- This model works well for smaller projects and projects where requirements are well understood.

Drawbacks of Classical Waterfall Model

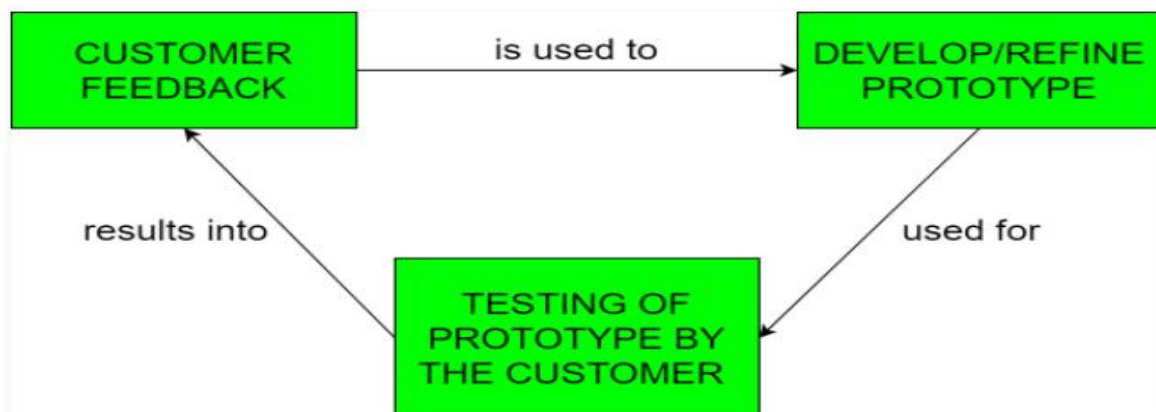
Classical waterfall model suffers from various shortcomings, basically we can't use it in real projects, but we use other software development lifecycle models which are based on the classical waterfall model. Below are some major drawbacks of this model:

1. Waterfall model cannot be used in real time projects. For example online related software's.
 2. Error correction not possible.
 3. Changing in requirement not possible.
 4. It is Phase dependent.
 5. It is time consuming.
 6. Some phases of this model can ideal
 7. Software development cost can be increased
 8. Software quality can be decreased.
- **No feedback path:** In classical waterfall model evolution of a software from one phase to another phase is like a waterfall. It assumes that no error is ever committed by developers during any phases. Therefore, it does not incorporate any mechanism for error correction.

- **Difficult to accommodate change requests:** This model assumes that all the customer requirements can be completely and correctly defined at the beginning of the project, but actually customers' requirements keep on changing with time. It is difficult to accommodate any change requests after the requirements specification phase is complete.
- **No overlapping of phases:** This model recommends that new phase can start only after the completion of the previous phase. But in real projects, this can't be maintained. To increase the efficiency and reduce the cost, phases may overlap.

Prototyping model:

Prototyping is defined as the process of developing a working replication of a product or system/software project that has to be engineered. It offers a small scale copy of the end product and is used for obtaining customer feedback as described below:



The Prototyping Model is one of the most popularly used Software Development Life Cycle Models (SDLC models). This model is used when the customers do not know the exact project requirements beforehand. In this model, a prototype of the end product is first developed, tested and refined as per customer feedback repeatedly till a final acceptable prototype is achieved which forms the basis for developing the final product.

In this process model, the system is partially implemented before or during the analysis phase thereby giving the customers an opportunity to see the product early in the life cycle.

The process starts by interviewing the customers and developing the incomplete high-level paper model. This document is used to build the initial prototype supporting only the basic functionality as desired by the customer. Once the customer figures out the problems, the prototype is further refined to

eliminate them. The process continues until the user approves the prototype and finds the working model to be satisfactory.

There are four types of model available:

A) Rapid Throwaway Prototyping –

This technique offers a useful method of exploring ideas and getting customer feedback for each of them. In this method, a developed prototype need not necessarily be a part of the ultimately accepted prototype. Customer feedback helps in preventing unnecessary design faults and hence, the final prototype developed is of better quality.

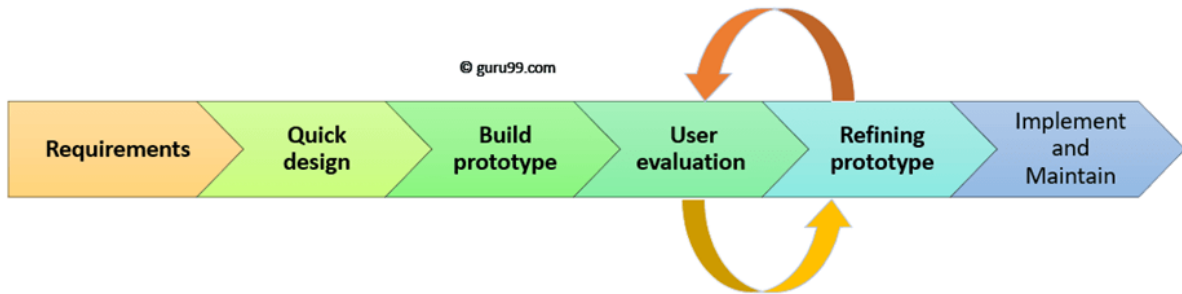
B) Evolutionary Prototyping –

In this method, the prototype developed initially is incrementally refined on the basis of customer feedback till it finally gets accepted. In comparison to Rapid Throwaway Prototyping, it offers a better approach which saves time as well as effort. This is because developing a prototype from scratch for every iteration of the process can sometimes be very frustrating for the developers.

C) Incremental Prototyping – In this type of incremental Prototyping, the final expected product is broken into different small pieces of prototypes and being developed individually. In the end, when all individual pieces are properly developed, then the different prototypes are collectively merged into a single final product in their predefined order. It's a very efficient approach which reduces the complexity of the development process, where the goal is divided into sub-parts and each sub-part is developed individually. The time interval between the project begin and final delivery is substantially reduced because all parts of the system are prototyped and tested simultaneously. Of course, there might be the possibility that the pieces just not fit together due to some lackness in the development phase – this can only be fixed by careful and complete plotting of the entire system before prototyping starts.

D) Extreme Prototyping – This method is mainly used for web development. It consists of three sequential independent phases:

Prototyping Model is a software development model in which prototype is built, tested, and reworked until an acceptable prototype is achieved. It also creates base to produce the final system or software. It works best in scenarios where the project's requirements are not known in detail. It is an iterative, trial and error method which takes place between developer and 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 their expectation from the system is.

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.

Step 6: Implement Product and Maintain

Once the final system is developed based on the final prototype, it is thoroughly tested and deployed to production. The system undergoes routine maintenance for minimizing downtime and prevent large-scale failures.

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 base lined.

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 consists of three sequential phases.

1. Basic prototype with all the existing page is present in the HTML format.
2. You can simulate data process using a prototype services layer.
3. The services are implemented and integrated into the final prototype.

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.

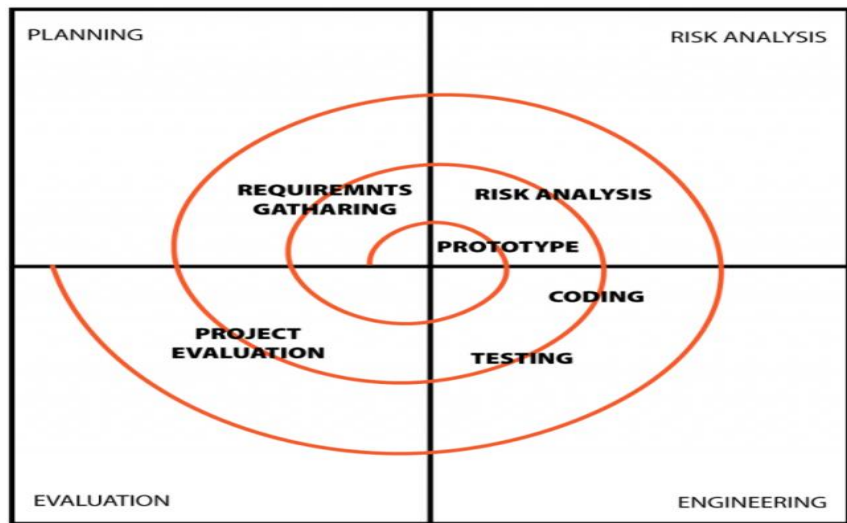
- Encourages innovation and flexible designing.
- It is a straightforward model, so it is easy to understand.
- No need for specialized experts to build the model
- The prototype serves as a basis for deriving a system specification.
- The prototype helps to gain a better understanding of the customer's needs.
- Prototypes can be changed and even discarded.
- A prototype also serves as the basis for operational specifications.
- Prototypes may offer early training for future users of the software system.

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.
- Sometimes 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.

Spiral model:



Spiral Model is a risk-driven software development process model.

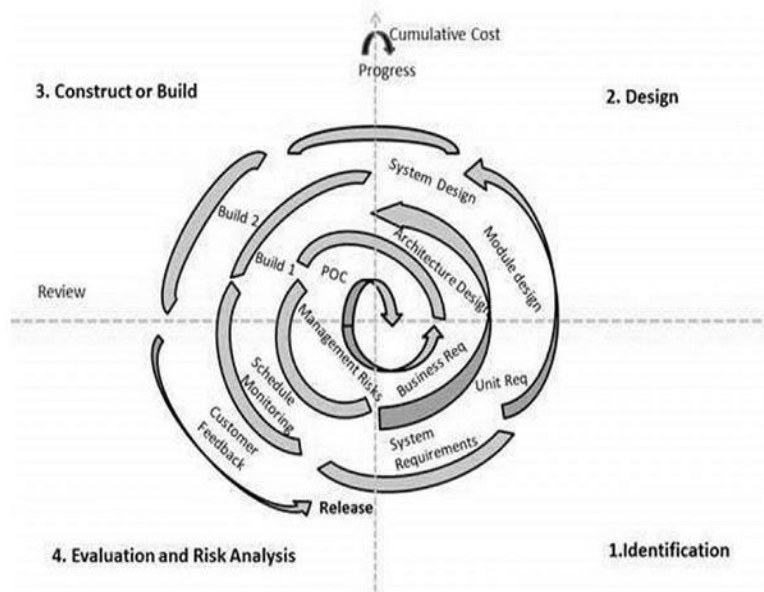
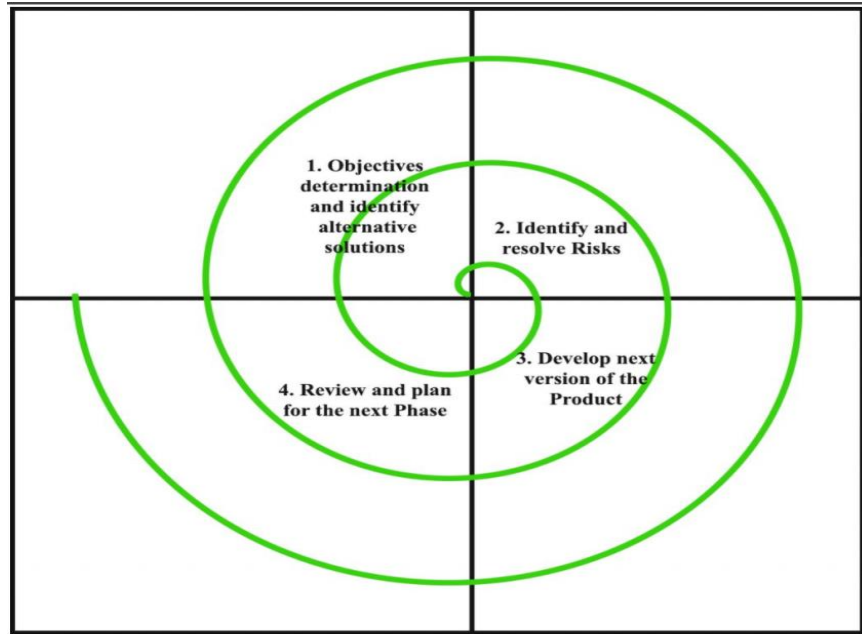
It is a combination of waterfall model and iterative model.

Spiral Model helps to adopt software development elements of multiple process models for the software project based on unique risk patterns ensuring efficient development process.

Each phase of spiral model in software engineering begins with a design goal and ends with the client reviewing the progress. The spiral model in software engineering was first mentioned by **Barry Boehm in his 1986 paper.**

The spiral model is a risk-driven software development process model. Based on the unique risk patterns of a given project, the spiral model guides a team to adopt elements of one or more process models, such as incremental, waterfall, or evolutionary prototyping.

The development process in Spiral model in SDLC, starts with a small set of requirement and goes through each development phase for those set of requirements. The software engineering team adds functionality for the additional requirement in every-increasing spirals until the application is ready for the production phase. The below figure very well explain Spiral Model:



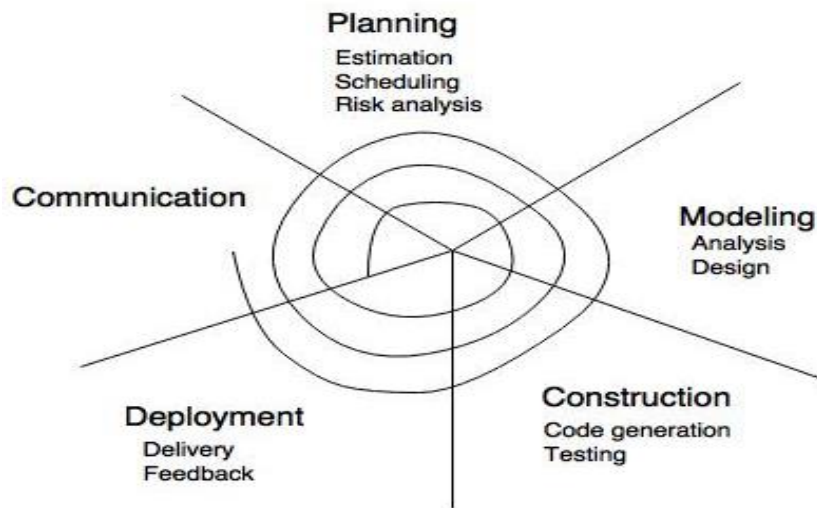


Fig. - The Spiral Model

Spiral Model Application/Uses

The Spiral Model is widely used in the software industry as it is in sync with the natural development process of any product, i.e. learning with maturity which involves minimum risk for the customer as well as the development firms.

The following pointers explain the typical uses of a Spiral Model –

- When there is a budget constraint and risk evaluation is important.
- For medium to high-risk projects.
- Long-term project commitment because of potential changes to economic priorities as the requirements change with time.
- Customer is not sure of their requirements which is usually the case.
- Requirements are complex and need evaluation to get clarity.
- New product line which should be released in phases to get enough customer feedback.
- Significant changes are expected in the product during the development cycle.

Spiral Model - Pros and Cons

The advantage of spiral lifecycle model is that it allows elements of the product to be added in, when they become available or known. This assures that there is no conflict with previous requirements and design.

This method is consistent with approaches that have multiple software builds and releases which allows making an orderly transition to a maintenance activity. Another

positive aspect of this method is that the spiral model forces an early User involvement in the system development effort.

On the other side, it takes a very strict management to complete such products and there is a risk of running the spiral in an indefinite loop. So, the discipline of change and the extent of taking change requests is very important to develop and deploy the product successfully.

The advantages of the Spiral SDLC Model are as follows –

- Changing requirements can be accommodated./ Changes can be accepted
- Allows extensive use of prototypes./reuse of previous requirements
- Requirements can be captured more accurately.
- Users see the system early.
- Development can be divided into smaller parts and the risky parts can be developed earlier which helps in better risk management.

The disadvantages of the Spiral SDLC Model are as follows –

- Management is more complex.
- End of the project may not be known early.
- Not suitable for small or low risk projects and could be expensive for small projects.
- Process is complex
- Spiral may go on indefinitely./due to changes adopt it require more time
- Large number of intermediate stages requires excessive documentation. / It requires more documentation.

Incremental model:

Explain incremental model briefly with a suitable example?

The **incremental build model** is a method of **software development** where the product is designed, implemented and tested **incrementally** (a little more is added each time) until the product is finished.

The product is defined as finished when it satisfies all of its requirements.

Incremental Model is a process of software development where requirements divided into multiple standalone modules of the software development cycle.

In this model, each module goes through the requirements, design, implementation and testing phases.

Every subsequent release of the module adds function to the previous release. The process continues until the complete system achieved.

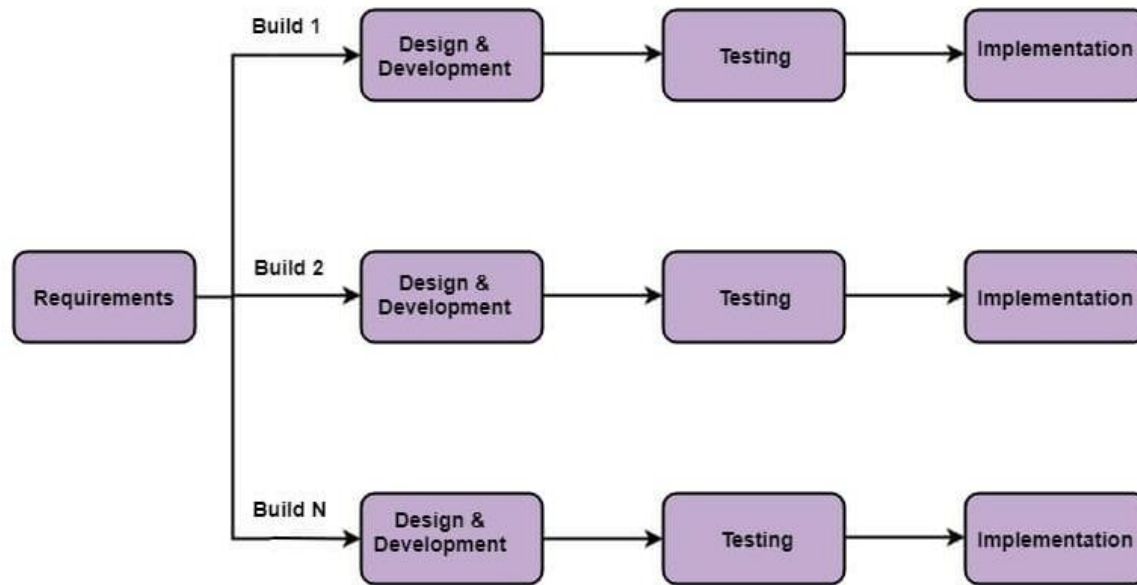


Fig: Incremental Model

The various phases of incremental model are as follows:

1. Requirement analysis: In the first phase of the incremental model, the product analysis expertise identifies the requirements. And the system functional requirements are understood by the requirement analysis team. To develop the software under the incremental model, this phase performs a crucial role.

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

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

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

When we use the Incremental Model?

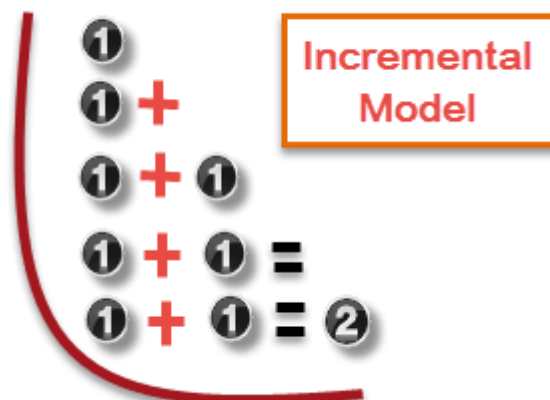
- When the requirements are superior.(conform, clear, well understood)
- A project has a lengthy development schedule. (when project size is large, when schedules have more time)
- When Software team are not very well skilled or trained.
- When the customer demands a quick release of the product.
- You can develop arranged requirements first.

Advantage of Incremental Model

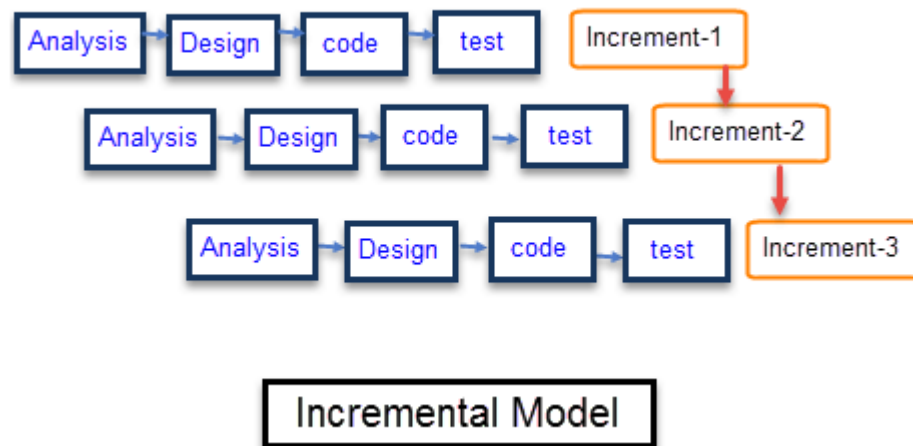
- Errors are easy to be recognized.
- Easier to test and debug
- More flexible.
- Simple to manage risk because it handled during its iteration.
- The Client gets important functionality early.

Disadvantage of Incremental Model

- Need for good planning/ it require the best planning.
- Total Cost is high. / it is costly.
- Well defined module interfaces are needed.



Each iteration passes through the **requirements, design, coding and testing phases**. And each subsequent release of the system adds function to the previous release until all designed functionality has been implemented.



The system is put into production when the first increment is delivered.

The first increment is often a core product where the basic requirements are addressed, and supplementary features are added in the next increments.

Once the core product is analyzed by the client, there is plan development for the next increment.

Characteristics of an Incremental module includes

- System/software development is broken down into many mini development projects
- Partial systems are successively built to produce a final total system
- Highest priority requirement is tackled first
- Once the requirement is developed, requirement for that increment are fixed.

Incremental Phases	Activities performed in incremental phases
Requirement Analysis	<ul style="list-style-type: none"> Requirement and specification of the software are collected
Design	<ul style="list-style-type: none"> Some high-end function are designed during this stage
Code	<ul style="list-style-type: none"> Coding of software is done during this stage
Test	<ul style="list-style-type: none"> Once the system is deployed, it goes through the testing phase

Advantages and Disadvantages of Incremental Model

Advantages	Disadvantages
<ul style="list-style-type: none"> The software will be generated quickly during the software life cycle 	<ul style="list-style-type: none"> It requires a good planning designing
<ul style="list-style-type: none"> It is flexible and less expensive to change requirements and scope 	<ul style="list-style-type: none"> Problems might cause due to system architecture as such not all requirements collected up front for the entire software lifecycle
<ul style="list-style-type: none"> Throughout the development stages changes can be done 	<ul style="list-style-type: none"> Each iteration phase is rigid and does not overlap each other
<ul style="list-style-type: none"> This model is less costly compared to others 	<ul style="list-style-type: none"> Rectifying a problem in one unit requires correction in all the units and consumes a lot of time
<ul style="list-style-type: none"> A customer can respond to each building 	
<ul style="list-style-type: none"> Errors are easy to be identified 	

Concurrent model (operating or occurring at the same time, running parallel):

The **concurrent development model**, sometimes called **concurrent engineering**,

can be represented as a series of framework activities, **Software engineering** actions of tasks, and their associated states. ... All activities exist **concurrently** but reside in different states.

Concurrent models are those **models** within which the various activities of software development happen at the same time, for faster development and a better outcome. The **concurrent model** is also referred to as a parallel working **model**.

- The concurrent development model is called as concurrent model.
- The communication activity has completed in the first iteration and exits in the awaiting changes state.
- The modeling activity completed its initial communication and then go to the underdevelopment state.
- If the customer specifies the change in the requirement, then the modeling activity moves from the under development state into the awaiting change state.
- The concurrent process model activities moving from one state to another state.

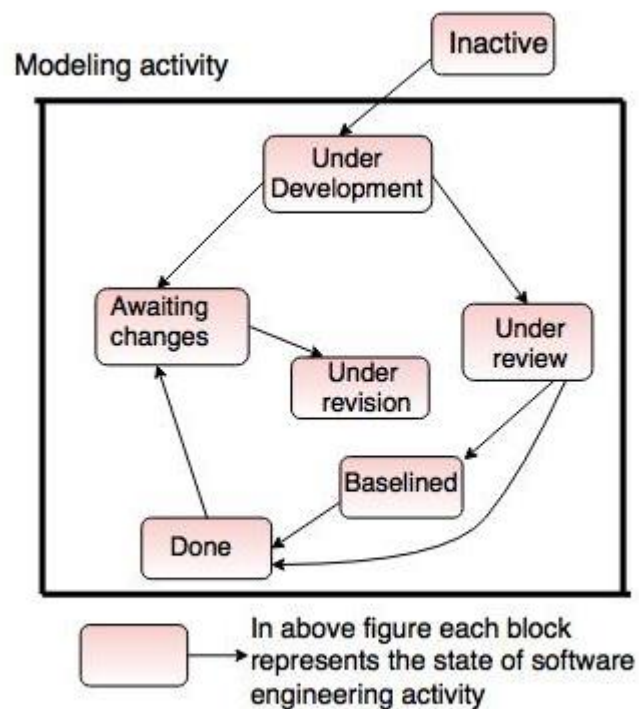


Fig. - One element of the concurrent process model

Advantages of the concurrent development model

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

- It is easy for understanding and use.
- It gives immediate feedback from testing.
- It provides an accurate picture of the current state of a project.

Disadvantages of the concurrent development model

- It needs better communication between the team members.
- This may not be achieved all the time.
- It requires to remember the status of the different activities.

• The concurrent development model, sometimes called concurrent engineering.

- It allows a software team to represent iterative and concurrent elements of any of the process model.

- For example, the modeling activity defined for the spiral model is accomplished by invoking one or more of the software engineering actions: prototyping, analysis, and design.

- The activity—modeling—may be in any one of the states noted at any given time.

- Similarly, other activities, actions, or tasks (e.g., communication or construction) can be represented in a similar manner.

- **All software engineering activities exist concurrently but reside in different states.**

Project Management Concepts:

What is Project?

A project is a group of tasks that need to complete to reach a clear result.

A project also defines as a set of inputs and outputs which are required to achieve a goal.

Projects can vary from simple to difficult and can be operated by one person or a hundred.

Projects usually described and approved by a project manager or team executive.

They go beyond their expectations and objects, and it's up to the team to handle logistics and complete the project on time.

For good project development, some teams split the project into specific tasks so they can manage responsibility and utilize team strengths.

What is software project management?

Software project management is an art and discipline of planning and supervising software projects.

It is a sub-discipline of software project management in which software projects planned, implemented, monitored and controlled.

What is Project?

A project is a group of tasks that need to complete to reach a clear result. A project also defines as a set of inputs and outputs which are required to achieve a goal.

Projects can vary from simple to difficult and can be operated by one person or a hundred.

Projects usually described and approved by a project manager or team executive.

They go beyond their expectations and objects, and it's up to the team to handle logistics and complete the project on time. For good project development, some teams split the project into specific tasks so they can manage responsibility and utilize team strengths.

What is software project management?

Software project management is an art and discipline of planning and supervising software projects. It is a sub-discipline of software project management in which software projects planned, implemented, monitored and controlled.

It is a procedure of managing, allocating and timing resources to develop computer software that fulfills requirements.

In software Project Management, the client and the developers need to know the length, period and cost of the project.

Requirement of software project management?

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

1. Time
2. Cost
3. Quality

It is an essential part of the software organization to deliver a quality product, keeping the cost within the client's budget and deliver the project as per schedule. There are various factors, both external and internal, which may impact this triple factor. Any of three-factor can severely affect the other two.

Project Manager

A project manager is a character who has the overall responsibility for the planning, design, execution, monitoring, controlling and closure of a project.

A project manager represents an essential role in the achievement of the projects.

A project manager is a character who is responsible for giving decisions, both large and small projects.

The project manager is used to manage the risk and minimize uncertainty. Every decision the project manager makes must directly profit their project.

Role of a Project Manager:

1. Leader

A project manager must lead his team and should provide them direction to make them understand what is expected from all of them.

2. Medium:

The Project manager is a medium between his clients and his team. He must coordinate and transfer all the appropriate information from the clients to his team and report to the senior management.

3. Mentor:

He should be there to guide his team at each step and make sure that the team has an attachment. He provides a recommendation to his team and points them in the right direction.

Responsibilities of a Project Manager:

1. Managing risks and issues.
2. Create the project team and assigns tasks to several team members.
3. Activity planning and sequencing.
4. Monitoring and reporting progress.
5. Modifies the project plan to deal with the situation.

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.

Need of 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 client's requirements. The most important is that the basic technology changes and advances so frequently and rapidly that experience of one product may not be applied to the other one. Such type 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 product, keeping the cost within client's budget constrain and deliver the project as per scheduled.

Hence in order, software project management is necessary to incorporate user requirements along with budget and time constraints.

Software Project Management consists of several different type of managements:

1. Conflict Management:

Conflict management is the process to restrict the negative features of

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

2. Risk Management:

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

3. Requirement Management:

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

4. Change Management:

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

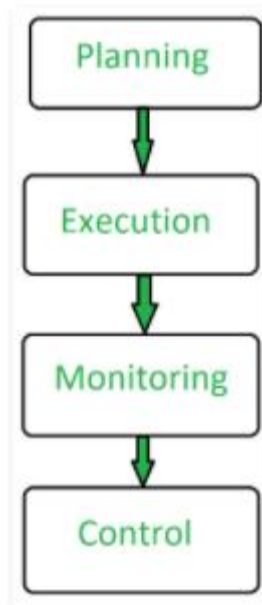
5. Software Configuration Management:

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

6. Release Management:

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

Aspects of Software Project Management:



Advantages of Software Project Management:

- It helps in planning of software development.
- Implementation of software development is made easy.
- Monitoring and controlling are aspects of software project management.
- It overall manages to save time and cost for software development.

The management spectrum :(Management Spectrum focuses on for P's)

In software engineering, the management spectrum describes the management of a software project.

The management of a software project starts from requirement analysis and finishes based on the nature of the product, it may or may not end because almost all software products faces changes and requires support.

It is about turning the project from plan to reality.

The management spectrum focuses on the four P's; people, product, process and project.

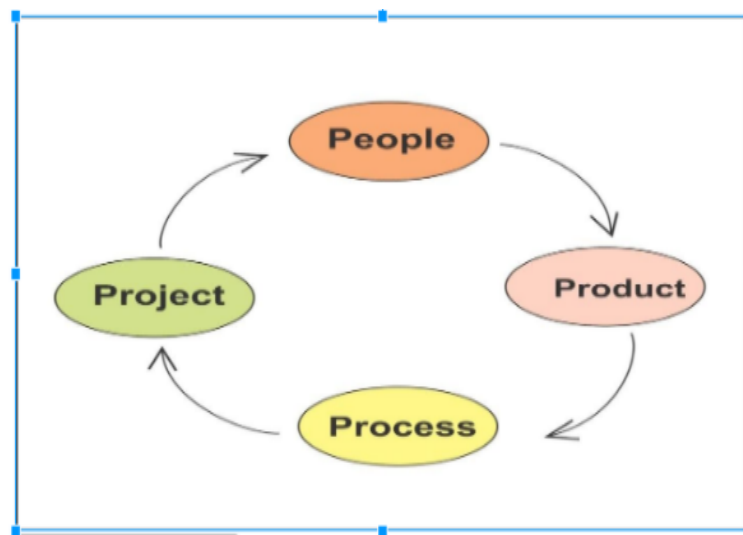
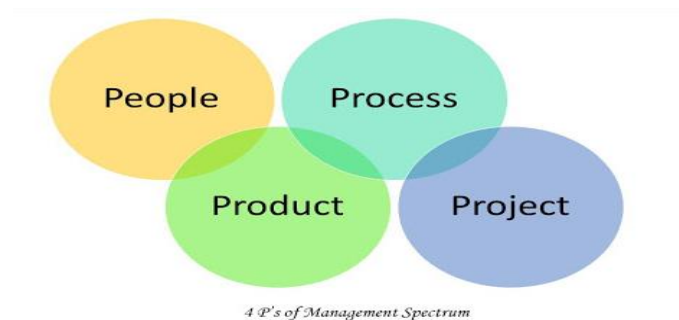
Here, the manager of the project has to control all these P's to have a smooth flow in the progress of the project and to reach the goal.

The management spectrum describes the management of a software project or how to make a project successful.

It focuses on the four P's; people, product, process and project. Here, the manager of the project has to control all these P's to have a smooth flow in the project progress and to reach the goal.

The four P's of management spectrum are...

1. People
2. Product
3. Process
4. Project



The four P's of management spectrum has been described briefly in below.

The People:

“Every organization needs to continually improve its ability to attract, develop, motivate, organize, and retain the workforce needed to accomplish its strategic business objectives”. The people capability maturity model defines the following key practice areas for software people: staffing, communication and coordination, work environment, performance management, training, compensation, competency analysis and development, career development, workgroup development, team/culture development, and others.

People of a project includes from manager to developer, from customer to end user. But mainly people of a project highlight the developers. It is so important to have highly skilled and motivated developers that the Software Engineering Institute has developed a People Management Capability Maturity Model (PM-CMM), “to enhance the readiness of software organizations to undertake increasingly complex applications by helping to attract, grow, motivate, deploy, and retain the talent needed to improve their software development capability”. Organizations that achieve high levels of maturity in the people management area have a higher likelihood of implementing effective software engineering practices.

The Product:

Product is any software that has to be developed. To develop successfully, product objectives and scope should be established, alternative solutions should be considered, and technical and management constraints should be identified. Without this information, it is impossible to define reasonable and accurate estimates of the cost, an effective assessment of risk, a realistic breakdown of project tasks or a manageable project schedule that provides a meaningful indication of progress.

The product is the ultimate goal of the project. This is any types of software product that has to be developed. To develop a software product successfully, all the product objectives and scopes should be established, alternative solutions should be considered, and technical and management constraints should be identified beforehand. Lack of these information, it is impossible to define reasonable and accurate estimation of the cost, an effective assessment of risks, a realistic breakdown of project tasks or a manageable project schedule that provides a meaningful indication of progress.

The Process:

(Process is a set of activities which is perform to complete a particular task)

A software process provides the framework from which a comprehensive plan for software development can be established.

A number of different tasks sets— tasks, milestones, work products, and quality assurance points—enable the framework activities to be adapted to the characteristics of the software project and the requirements of the project team.

Finally, umbrella activities overlay the process model. Umbrella activities are independent of any one framework activity and occur throughout the process.

The Project:

Here, the manager has to do some job. The project includes all and everything of the total development process and to avoid project failure the manager has to take some steps, has to be concerned about some common warnings etc.

The project is the complete software project that includes requirement analysis, development, delivery, maintenance and updates. The project manager of a project or sub-project is responsible for managing the people, product and process. The responsibilities or activities of software project manager would be a long list but that has to be followed to avoid project failure.

A software project could be extremely complex and as per the industry data the failure rate is high. Its merely due to the development but mostly due to the steps before development and sometimes due to the lack of maintenance.

UNIT II: METRICS

Software Process and Project Metrics:

Measures:

Software Measurement: A measurement is an appearance of the size, quantity, amount or dimension of a particular attributes of a product or process.

For developing a particular software product how much efforts we need and apply all that are counting and this counting is called as measures or software measure.

Software measurement is a titrate attribute of a characteristic of a software product or the software process. It is an authority within software engineering. Software measurement process is defined and governed by ISO Standard.

Need of Software Measurement:

Software is measured to:

1. Create the quality of the current product or process.
2. Anticipate future qualities of the product or process.
3. Improve the quality of a product or process.
4. Regulate the state of the project in relation to budget and schedule.

Classification of Software Measurement:

There are 2 types of software measurement:

1. Direct Measurement:
In direct measurement the product, process or thing is measured directly using standard scale.
2. Indirect Measurement:
In indirect measurement the quantity or quality to be measured is measured using related parameter i.e. by use of reference.

For example Software size, functional measurement, measuring code, Measuring software complexity, organization and link.

Measurement is the numerical quantification of the attributes of an object or event, which can be used to compare with other objects or events.

To assess (Examines) the quality of the engineered product or system and to better understand the models that are created, some measures are used.

These measures are collected throughout the software development life cycle with an intention to improve the software process on a continuous basis.

Measurement helps in estimation, quality control, productivity assessment and project control throughout a software project.

Also, measurement is used by software engineers to gain insight into the design and development of the work products. In addition, measurement assists in strategic decision-making as a project proceeds.

Software measurements are of two categories, namely,

1. Direct measures and indirect measures. Direct measures include software processes like cost and effort applied and products like lines of code produced, execution speed, and other defects that have been reported.
2. Indirect measures include products like functionality, quality, complexity, reliability, maintainability, and many more.

Generally, software measurement is considered as a management tool which if conducted in an effective manner, helps the project manager and the entire software team to take decisions that lead to successful completion of the project. Measurement process is characterized by a set of five activities, which are listed below.

1. Formulation: This performs measurement and develops appropriate metric for software under consideration.
2. Collection: This collects data to derive the formulated metrics.
3. Analysis: This calculates metrics and the use of mathematical tools.
4. Interpretation: This analyzes the metrics to attain insight into the quality of representation.
5. Feedback: This communicates recommendation derived from product metrics to the software team.

Note that collection and analysis activities drive the measurement process.

In order to perform these activities effectively, it is recommended to automate data collection and analysis, establish guidelines and recommendations for each metric, and use statistical techniques to interrelate external quality features and internal product attributes.

- 1. Explain software measurement process briefly.**
- 2. What mean by software measurement.**
- 3. What are software metrics and measurement.**

Metrics and indicators:

A software metric is a measure of software characteristics which are measurable or countable. Software metrics are valuable for many reasons, including measuring software performance, planning work items, measuring productivity, and many other uses.

Within the software development process, many metrics are that are all connected. Software metrics are similar to the four functions of management: Planning, Organization, Control, or Improvement.

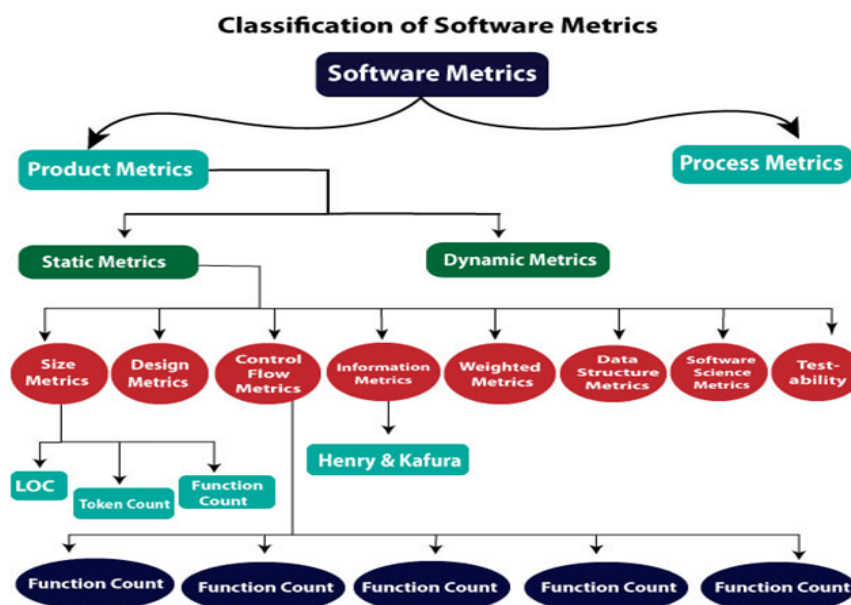
Software metrics can be classified into two types as follows:

1. Product Metrics: These are the measures of various characteristics of the software product. The two important software characteristics are:

1. Size and complexity of software.
2. Quality and reliability of software.

These metrics can be computed for different stages of SDLC.

2. Process Metrics: These are the measures of various characteristics of the software development process. For example, the efficiency of fault detection. They are used to measure the characteristics of methods, techniques, and tools that are used for developing software.



Internal metrics: Internal metrics are the metrics used for measuring properties that are viewed to be of greater importance to a software developer. For example, Lines of Code (LOC) measure.

External metrics: External metrics are the metrics used for measuring properties that are viewed to be of greater importance to the user, e.g., portability, reliability, functionality, usability, etc.

Hybrid metrics: Hybrid metrics are the metrics that combine product, process, and resource metrics. For example, cost per FP where FP stands for Function Point Metric.

Project metrics: Project metrics are the metrics used by the project manager to check the project's progress. Data from the past projects are used to collect various metrics, like time and cost; these estimates are used as a base of new software. Note that as the project proceeds, the project manager will check its progress from time-to-time and will compare the effort, cost, and time with the original effort, cost and time. Also understand that these metrics are used to decrease the development costs, time efforts and risks. The project quality can also be improved. As quality improves, the number of errors and time, as well as cost required, is also reduced.

Advantage of Software Metrics

Comparative study of various design methodology of software systems.

For analysis, comparison, and critical study of different programming language concerning their characteristics.

In comparing and evaluating the capabilities and productivity of people involved in software development.

In the preparation of software quality specifications.

In the verification of compliance of software systems requirements and specifications.

In making inference about the effort to be put in the design and development of the software systems.

In getting an idea about the complexity of the code.

In taking decisions regarding further division of a complex module is to be done or not.

In guiding resource manager for their proper utilization.

In comparison and making design tradeoffs between software development and maintenance cost.

In providing feedback to software managers about the progress and quality during various phases of the software development life cycle.

In the allocation of testing resources for testing the code.

Disadvantage of Software Metrics

The application of software metrics is not always easy, and in some cases, it is difficult and costly.

The verification and justification of software metrics are based on historical/empirical data whose validity is difficult to verify.

These are useful for managing software products but not for evaluating the performance of the technical staff.

The definition and derivation of Software metrics are usually based on assuming which are not standardized and may depend upon tools available and working environment.

Most of the predictive models rely on estimates of certain variables which are often not known precisely.

Indicators:

Indicators are calculated measures of performance consisting of a set of different metrics. They are also called Key Performance Indicators or simply KPIs.

Because they are a bit more complex data (and may contain more than one variable), performance indicators can provide a more accurate view of a situation and its historical evolution. They work as clues as to whether everything is going well and whether the desired goals have been achieved.

KPIs evaluate organizational performance, assist in trend analysis, promote continuous improvement and proactive performance, besides transparent management of processes and staff. They are usually expressed in percent rate or frequency formats.

An indicator is a qualitative or quantitative factor or variable that provides a simple and reliable mean to express achievement, the attainment of a goal, or the results stemming from a specific change. It often aggregates or combines multiple measures in an explicit formula.

1M weekly active users

1:3 users complete the story

23% homepage bounce rate

All indicators are relative to a set of measures in a particular context. For an indicator to be meaningful, it should always be contextualized with the scope of stories being assessed and the measures used to calculate should be made explicit.

It is very easy to get indicators wrong because they can obscure a lot of information and complexity with the deceptively simple guise of a single value. Here are some situations:

1. Indicators are always relative to a context

2. Indicators can be simplifying to a fault

3. Indicators force value judgments

Difference in Measures, Metrics, and Indicators

Metrics is often used interchangeably with measure and measurement. However, it is important to note the differences between them. Measure can be defined as quantitative indication of amount, dimension, capacity, or size of product and process attributes. Measurement can be defined as the process of determining the measure. Metrics can be defined as quantitative measures that allow software engineers to identify the efficiency and improve the quality of software process, project, and product.

To understand the difference, let us consider an example. A measure is established when a number of errors is (single data point) detected in a software component. Measurement is the process of collecting one or more data points. In other words, measurement is established when many components are reviewed and tested individually to collect the measure of a number of errors in all these components. Metrics are associated with individual measure in some manner. That is, metrics are related to detection of errors found per review or the average number of errors found per unit test.

Once measures and metrics have been developed, indicators are obtained. These indicators provide a detailed insight into the software process, software project, or intermediate product. Indicators also enable software engineers or project managers to adjust software processes and improve software products, if required. For example, measurement dashboards or key indicators are used to monitor progress and initiate change. Arranged together, indicators provide snapshots of the system's performance.

Measured Data

Before data is collected and used, it is necessary to know the type of data involved in the software metrics. Table lists different types of data, which are identified in metrics along with their description and the possible operations that can be performed on them.

Type of Data Measured

Type of data	Possible operations	Description of data
Nominal	=, ≠	Categories
Ordinal	<, >	Ranking
Interval	+, −	Differences
Ratio	/	Absolute zero

- **Nominal data:** Data in the program can be measured by placing it under a category. This category of program can be a [database](#) program, application program, or an [operating system](#) program. For such data, operation of arithmetic type and ranking of values in any order (increasing or decreasing) is not possible. The only operation that can be performed is to determine whether program 'X' is the same as program 'Y'.
- **Ordinal data:** Data can be ranked according to the data values. For example, experience in application domain can be rated as very low, low, medium, or high. Thus, experience can easily be ranked according to its rating.
- **Interval data:** Data values can be ranked and substantial differences between them can also be shown. For example, a program with complexity level 8 is said to be 4 units more complex than a program with complexity level 4.
- **Ratio data:** Data values are associated with a ratio scale, which possesses an absolute zero and allows meaningful ratios to be calculated. For example, program lines expressed in lines of code.

It is desirable to know the measurement scale for metrics. For example, if metrics values are used to represent a model for a software process, then metrics associated with the ratio scale may be preferred.

Guidelines for Software Metrics

Although many software metrics have been proposed over a period of time, ideal software metric is the one which is easy to understand, effective, and efficient. In order to develop ideal metrics, software metrics should be validated and characterized effectively. For this, it is important to develop metrics using some specific guidelines, which are listed below.

- **Simple and computable:** Derivation of software metrics should be easy to learn and should involve average amount of time and effort.
- **Consistent and objective:** Unambiguous results should be delivered by software metrics.
- **Consistent in the use of units and dimensions:** Mathematical computation of the metrics should involve use of dimensions and units in a consistent manner.
- **Programming language independent:** Metrics should be developed on the basis of the analysis model, design model, or program's structure.
- **High quality:** Effective software metrics should lead to a high-quality software product.
- **Easy to calibrate:** Metrics should be easy to adapt according to project requirements.
- **Easy to obtain:** Metrics should be developed at a reasonable cost.
- **Validation:** Metrics should be validated before being used for making any decisions.

- **Robust:** Metrics should be relatively insensitive to small changes in process, project, or product.
- **Value:** Value of metrics should increase or decrease with the value of the software characteristics they represent. For this, the value of metrics should be within a meaningful range. For example, metrics can be in a range of 0 to 5.

Software measurement:

Software measurement is a quantified (Calculated) attribute (A measurement is an appearance of the size, quantity, amount or dimension of a particular attributes of a product or process.) It is a discipline within software engineering. The process of software measurement is defined and governed by ISO Standard ISO 15939 (software measurement process).

E.g. Software size, functional measurement, Measuring code, Measuring software complexity,

Size-oriented Metrics:

Size Oriented Metrics derived by normalizing quality and productivity Point Metrics measures by considering size of the software that has been produced. The organization builds a simple record of size measure for the software projects. It is built on past experiences of organizations. It is a direct measure of software.

This metrics is one of simplest and earliest metrics that is used for computer program to measure size. Size Oriented Metrics are also used for measuring and comparing productivity of programmers. It is a direct measure of a Software. The size measurement is based on lines of code computation. The lines of code are defined as one line of text in a source file.

While counting lines of code, simplest standard is:

- Don't count blank lines
- Don't count comments
- Count everything else

The size-oriented measure is not a universally accepted method. Simple set of size measure that can be developed is as given below:

```
Size = Kilo Lines of Code (KLOC)
Effort = Person / month
Productivity = KLOC / person-month
Quality = Number of faults / KLOC
Cost = $ / KLOC
Documentation = Pages of documentation / KLOC
```

Advantages:

- Using these metrics, it is very simple to measure size.
- Artifact of Software development which is easily counted.
- LOC is used by many methods that are already existing as a key input.
- A large body of literature and data based on LOC already exists.

Disadvantages:

- This measure is dependent upon programming language.
- This method is well designed upon programming language.
- It does not accommodate non-procedural languages.
- Sometimes, it is very difficult to estimate LOC in early stage of development.
- Though it is simple to measure but it is very hard to understand it for users.
- It cannot measure size of specification as it is defined on code.

Example –

For a size oriented metrics, software organization maintains records in tabular form. The typical table entries are: Project Name, LOC, Efforts, Pages of documents, Errors, Defects, Total number of people working on it.

Project Name	LOC	Effort	Cost (\$)	Doc. (pages)	Errors	Defects	People
ABC	10,000	20	170	400	100	12	4
PQR	20,000	60	300	1000	129	32	6
XYZ	35,000	65	522	1290	280	87	7

Size Oriented Metrics

LOC Metrics

It is one of the earliest and simpler metrics for calculating the size of the computer program. It is generally used in calculating and comparing the productivity of programmers. These metrics are derived by normalizing the quality and productivity measures by considering the size of the product as a metric.

Following are the points regarding LOC measures:

1. In size-oriented metrics, LOC is considered to be the normalization value.
2. It is an older method that was developed when FORTRAN and COBOL programming were very popular.
3. Productivity is defined as $KLOC / EFFORT$, where effort is measured in person-months.
4. Size-oriented metrics depend on the programming language used.
5. As productivity depends on KLOC, so assembly language code will have more productivity.
6. LOC measure requires a level of detail which may not be practically achievable.
7. The more expressive is the programming language, the lower is the productivity.
8. LOC method of measurement does not apply to projects that deal with visual (GUI-based) programming. As already explained, Graphical User Interfaces (GUIs) use forms basically. LOC metric is not applicable here.
9. It requires that all organizations must use the same method for counting LOC. This is so because some organizations use only executable statements, some useful comments, and some do not. Thus, the standard needs to be established.
10. These metrics are not universally accepted.

Based on the LOC/KLOC count of software, many other metrics can be computed:

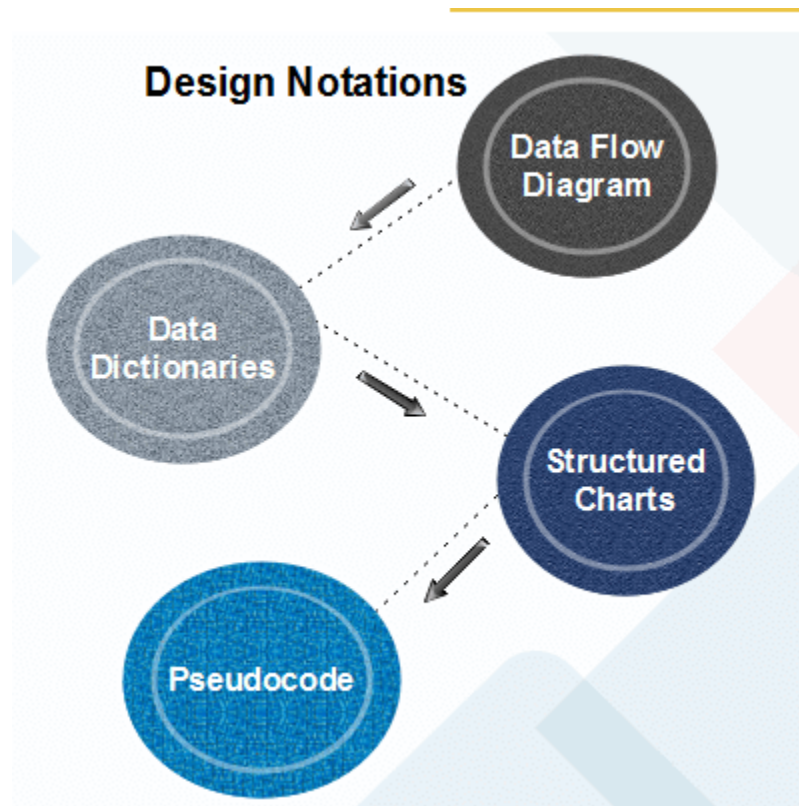
- a. Errors/KLOC.
- b. \$/ KLOC.
- c. Defects/KLOC.
- d. Pages of documentation/KLOC.
- e. Errors/PM.
- f. Productivity = $KLOC/PM$ (effort is measured in person-months).
- g. \$/ Page of documentation.

Function - oriented metrics:

Function Oriented design is a method to software design where the model is decomposed into a set of interacting units or modules where each unit or module has a clearly defined function. Thus, the system is designed from a functional viewpoint.

Design Notations

Design Notations are primarily meant to be used during the process of design and are used to represent design or design decisions. For a function-oriented design, the design can be represented graphically or mathematically by the following:







Data Flow Diagram

Data-flow design is concerned with designing a series of functional transformations that convert system inputs into the required outputs. The design is described as data-flow diagrams. These diagrams show how data flows through a system and how the output is derived from the input through a series of functional transformations.

Data-flow diagrams are a useful and intuitive way of describing a system. They are generally understandable without specialized training, notably if control information is excluded. They show end-to-end processing. That is the flow of processing from when data enters the system to where it leaves the system can be traced.

Data-flow design is an integral part of several design methods, and most CASE tools support data-flow diagram creation. Different ways may use different icons to represent data-flow diagram entities, but their meanings are similar.

The notation which is used is based on the following symbols:

Symbol	Name	Meaning
	Rounded Rectangle	It represents functions which transforms input to output. The transformation name indicates its function.
	Rectangle	It represents data stores. Again, they should give a descriptive name.
	Circle	It represents user interactions with the system that provides input or receives output.
	Arrows	It shows the direction of data flow. Their name describes the data flowing along the path.
"and" and "or"	Keywords	The keywords "and" and "or". These have their usual meanings in boolean expressions. They are used to link data flows when more than one data flow may be input or output from a transformation.

Function-Oriented Metrics is a method that is developed by Albrecht in 1979 for IBM (International Business Machine). He simply suggested a measure known as Function points that are derived using an empirical relationship that is based on countable measures of software's information or requirements domain and assessments of the complexity of software.

Function-Oriented Metrics are also known as **Function Point Model**. This model generally focuses on the functionality of the software application being delivered. These methods are actually independent of the programming language that is being used in software applications and based on calculating the Function Point (FP). A function point is a unit of measurement that measures the business functionality provided by the business product.

To determine whether or not a particular entry is simple, easy, average, or complex, a criterion is needed and should be developed by the organization. With the help of observations or experiments, the different weighing factors should be determined as shown below in the table. With the help of these tables, the count table can be computed.

Function-oriented software metrics use a measure of the functionality delivered by the application as a normalization value. Since ‘functionality’ cannot be measured directly, it must be derived indirectly using other direct measures. Function-oriented metrics were first proposed by Albrecht, who suggested a measure called the function point. Function points are derived using an empirical relationship based on countable (direct) measures of software's information domain and assessments of software complexity.

DOMAIN CHARACTERISTICS	COUNT		WEIGHTING FACTOR			COUNT
			SIMPLE	AVERAGE	COMPLEX	
Number of User Input		X	3	4	6	
Number of User Output		X	4	5	7	
Number of User Inquiries		X	3	4	6	
Number of Files		X	7	10	15	
Number of External Interfaces		X	5	7	10	
Count total						

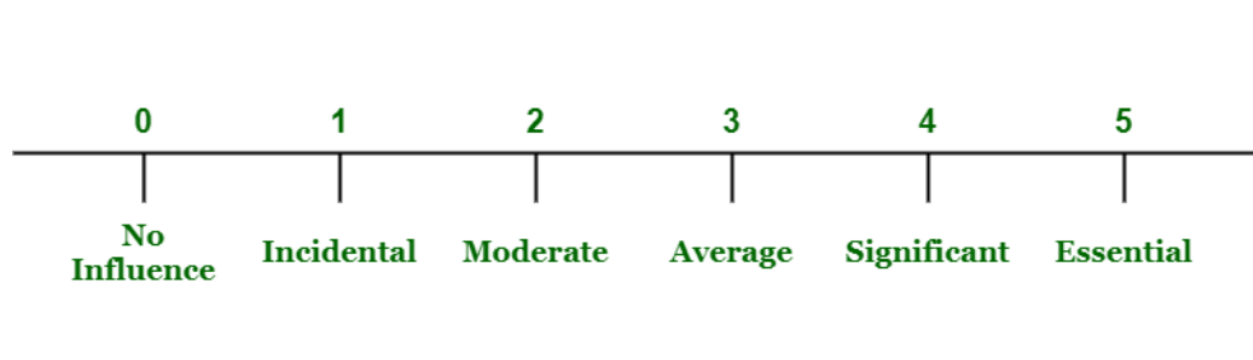
The software complexity can be computed by answering the following questions :

- Does the system need reliable backup and recovery?
- Are data communications required?
- Are there distribute processing functions?
- Is the performance of the system critical?
- Can the system be able to run in an existing, heavily, and largely utilized operational environment?
- Does the system require on-line data entry?
- Does the input transaction is required by the on-line data entry to be built over multiple screens or operations?
- Are the master files updated on-line?
- Are the inputs, outputs, files, or inquiries complex?
- Is the internal processing complex?
- Is the code which is designed to be reusable?
- Are conversion and installation included in the design?
- Is the system designed for multiple installations in various organizations whenever required?

- Is the application designed to facilitate or make the change and provide effective ease of use by the user?

Each of the above questions is answered using a scale that ranges from 0 to 5 (not important or applicable to absolutely essential).

This scale is shown below :



Calculating Function Point :

After calculating the function point, various other measures can be calculated as shown below :

```
Productivity = FP / person-month  
Quality = Number of faults / FP  
Cost = $ / FP  
Documentation = Pages of documentation / FP
```

Some figures –

Language	Lines of Code per Function Point
ADA 83	71
C	128
C++	49
CLOS	27
COBOL 85	91
Eiffel	21
C++	21
Smalltalk	21
Visual Basic	32

Small Project: <2000 Function Points

Medium Project: 2, 000 to 10, 000 Function Points

Large Project: > 10, 000 Function Points

Disadvantages of Function-Oriented Metrics:

- Function Oriented Metrics was only developed for business systems, therefore it is valid for only that domain.
- In this, some of the aspects are subjective and have not been validated.
- The function point does not have any physical meaning. It is just a number.

Extended function point metrics:

Function Point (FP) measure was insufficient for many engineering and embedded systems. To overcome this, A number of extensions to the basic function point measure have been proposed. These are as follows:

Feature Points:

- Feature Points are computed by counting the information domain values.
- It can be used in those areas where there is a level of complexity, is comparatively very high.
- Function point (FP) measure is the subset for the Feature point.
- But both the Function point and feature point represents the functionality of the systems

3D function points:

- Data, Functional, and control are three dimensions represented by 3D function points.
 1. **Data:** User interfaces and data as in the original method.
 2. **Control:** Real-time behavior(s)
 3. **Function:** Internal processing
- Data dimension calculation is the same as the FPs. Feature-Transformation is done in the functional dimension. While in the control dimension, feature-Transition is added.
- The 3D Function Point method was proposed by Boeing.
- It is designed to solve two problems with the Albrecht approach.

Example:

Compute the FP, feature point and 3D-function point value for an embedded system with the following characteristics:

1. Internal data structures = 8
2. No. of user inputs = 32
3. No. of user outputs = 60
4. No. of user inquiries = 24
5. No. of external interfaces = 2
6. No. of transformation = 23
7. No. of transition = 32

The function point measure was originally designed to be applied to business information systems applications. To accommodate these applications, the data dimension was emphasized to the exclusion of the functional and behavioral (control) dimensions. For this reason, the function point measure was inadequate for many engineering and embedded systems (which emphasize function and control). A number of extensions to the basic function point measure have been proposed to remedy this situation.

A function point extension called feature points, is a superset of the function point measure that can be applied to systems and engineering software applications. The feature point measure accommodates applications in which algorithmic complexity is high. Real-time, process control and

embedded software applications tend to have high algorithmic complexity and are therefore amenable to the feature point.

To compute the feature point, information domain values are again counted and weighted. In addition, the feature point metric counts a new software characteristic—algorithms. An algorithm is defined as "a bounded computational problem that is included within a specific computer program". Inverting a matrix, decoding a bit string, or handling an interrupt are all examples of algorithms.

Another function point extension for real-time systems and engineered products has been developed by Boeing. The Boeing approach integrates the data dimension of software with the functional and control dimensions to provide a function-oriented measure amenable to applications that emphasize function and control capabilities. Called the 3D function point, characteristics of all three software dimensions are "counted, quantified, and transformed" into a measure that provides an indication of the functionality delivered by the software. The data dimension is evaluated in much the same way as described before. Counts of retained data (the internal program data structure; e.g., files) and external data (inputs, outputs, inquiries, and external references) are used along with measures of complexity to derive a data dimension count. The functional dimension is measured by considering "the number of internal operations required to transform input to output data". For the purposes of 3D function point computation, a "transformation" is viewed as a series of processing steps that are constrained by a set of semantic statements. The control dimension is measured by counting the number of transitions between states.

A state represents some externally observable mode of behavior, and a transition occurs as a result of some event that causes the software or system to change its mode of behavior (i.e., to change state). For example, a wireless phone contains software that supports auto dial functions. To enter the auto-dial state from a resting state, the user presses an Auto key on the keypad. This event causes an LCD display to prompt for a code that will indicate the party to be called. Upon entry of the code and hitting the Dial key (another event), the wireless phone software makes a transition to the dialing state. When computing 3D function points, transitions are not assigned a complexity value.

4. Software Project Planning:

Project planning objectives:

Project Planning Objectives:

The objective of software project planning is to provide a framework that enables the Manager to make reasonable estimates of:

- 1. Resources**
- 2. Cost, and**
- 3. Schedule**

These estimates are made within a limited time frame at the beginning of a software Project and should be updated regularly as the project progresses.

In addition, estimates should attempt to define best case and worst-case scenarios so That project outcomes can be bounded.

Planning is one of the most important management activities and is an ongoing effort Throughout the life of the project.

Software project management begins with a set of activities that are collectively Called Project Planning.

The software project planner must estimate following things before a project begins:

- 1. How much will it cost?**
- 2. How long will it take?**
- 3. How many people will it take?**
- 4. What might go wrong?**

Once a project is found to be possible, computer code project managers undertake project designing. Project designing is undertaken and completed even before any development activity starts. Project designing consists of subsequent essential activities:

Estimating the subsequent attributes of the project:

Project size:

What's going to be downside quality in terms of the trouble and time needed to develop the product?

Cost:

What proportion is it reaching to value to develop the project?

Duration:

However long is it reaching to want complete development?

Effort:

What proportion effort would be required?

The effectiveness of the following designing activities relies on the accuracy of those estimations.

planning force and alternative resources

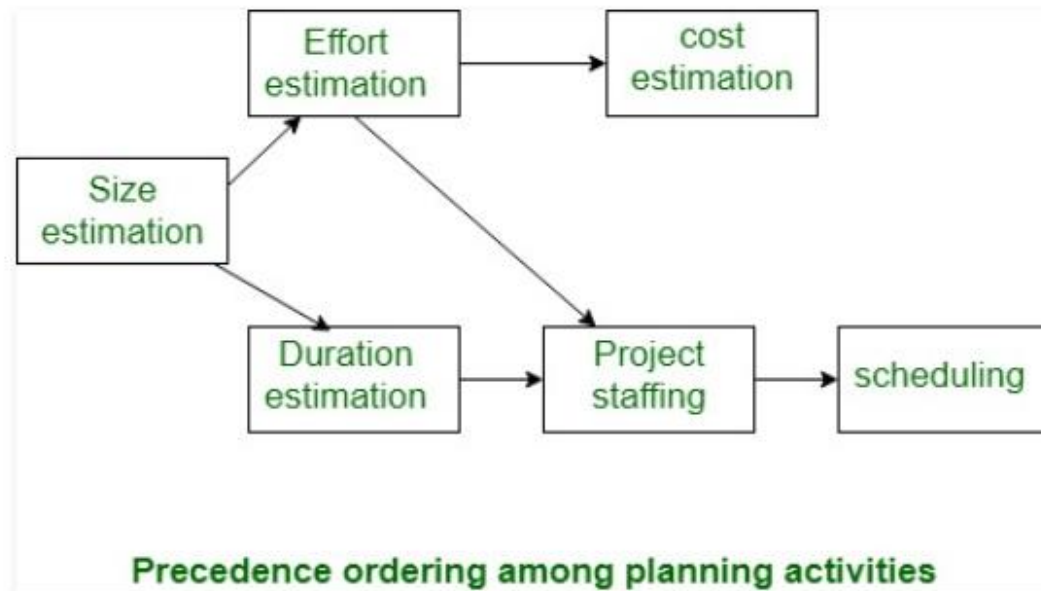
workers organization and staffing plans

Risk identification, analysis, and abatement designing

Miscellaneous arranges like quality assurance plan, configuration, management arrange, etc.

Precedence ordering among project planning activities:

The different project connected estimates done by a project manager have already been mentioned. The below diagram shows the order during which vital project coming up with activities is also undertaken. It may be simply discovered that size estimation is that the 1st activity. It's conjointly the foremost basic parameter supported that all alternative coming up with activities square measure dispensed, alternative estimations like the estimation of effort, cost, resource, and project length also are vital elements of the project coming up with.



Project Planning Objectives

- The objective of software project planning is to provide a framework that enables the project manager to make some reasonable estimates of resources, cost and schedule
- These estimates are made at the beginning of a software project and should be updated regularly as the project progresses towards completion
- The planning objective is achieved through a process of information discovery that leads to the formulation of reasonable estimates

Software Scope

- First activity in the planning of the software project is the determination of the scope of the software
- Scope of the software describes the data and control to be processed, function, performance, constraints, interfaces and reliability
- Obtaining information necessary for scope
- Feasibility

Resources

- The second task involved in software planning is the estimation of the resources required to accommodate the software development effort
- Project Development Resources
- Each resource is specified with 4 characteristics :

1. Description of the resource.

2. Statement of availability.
3. Time when the resource will be required.
4. Duration of time that resource will be applied

Software Project Estimation:

It have concern with pre-defining the cost of developing software product.

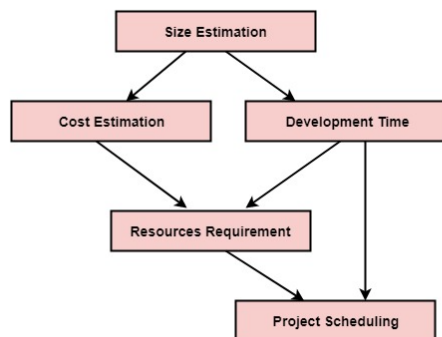
- Software is the most expensive element of virtually all computer-based systems
- Cost overrun can be disastrous for the software developer
- To achieve a reliable cost and effort estimates, a number of options arise :
 1. Delay estimation until late in the project.
 2. Base estimates on similar projects that have already been completed.
 3. Use relatively simple decomposition techniques to generate project cost and effort estimates.
 4. Use one or more empirical models for software cost and effort estimation.
 5. The first option, however attractive, is not practical
- the second option can work reasonably well, if the current project is quite similar to the past efforts and other project influences are equivalent
- Decomposing techniques take a divide and conquer approach to software project estimation
- Empirical estimation models can be used to complement decomposition techniques and offer a potentially valuable estimation approach in their own right

Decomposition Techniques:

The decomposition techniques are those techniques which can divide your software into several parts it is called as the decomposing tech.

- Software project estimation is a form of problem solving
- in most cases, the problem/software to be solved/create is too complex to be considered in one piece
- The decomposition approach can be seen from two different points of view
 - Decomposition of the problem.
 - Decomposition of the process.

Software Project planning starts before technical work start. The various steps of planning activities are:



The size is the crucial parameter for the estimation of other activities. Resources requirement are required based on cost and development time. Project schedule may prove to be very useful for controlling and monitoring the progress of the project. This is dependent on resources & development time.

Software project estimation:

Accurate estimations determine the overall success of a software project. They are essential for effective project planning and management.

Over-estimation of project effort may cause: under-utilized resources and a consequent cost blow-out. Future projects may be delayed due to the over-estimation of the current project duration.

Under-estimation causes tighter deadlines. It has been proven that adding more resources delays the project more. The rush to achieve project milestones can result in a loss of quality. Hence, more defects, when the project is implemented.

Estimation is a process to predict the time and the cost that a project requires to be finished appropriately. But in term of software development, it also means of the consideration of the experience of the software development company; the technique they employ; the process they may go through to finish the task. This whole process requires the use of complex tools and good mathematical background knowledge. It is in some cases is the accomplishment of hard work of a whole team. The error margin is, consequently, guaranteed possibly around 5-10%.

The whole process of estimation would cost the company rather considerable cost and time at the very first stage of building an app. But this will make the final result more credible, realistic, and customer-satisfying. Projects especially big ones are advisable to employ this crucial step to avoid unpredictable failure.

Estimation is the process of finding an estimate, or approximation, which is a value that can be used for some purpose even if input data may be incomplete, uncertain, or unstable.

Estimation determines how much money, effort, resources, and time it will take to build a specific system or product. Estimation is based on –

- Past Data/Past Experience
- Available Documents/Knowledge
- Assumptions
- Identified Risks

The four basic steps in Software Project Estimation are –

- Estimate the size of the development product.
- Estimate the effort in person-months or person-hours.
- Estimate the schedule in calendar months.
- Estimate the project cost in agreed currency.

Decomposition techniques:

Decomposition techniques works on “Divide and Conquer” approach in software project estimation. By decomposition a project is divided into different components and related software engineering activities. Cost and effort estimation can be performed step by step on each component.

Software cost estimation is a form to solve the problems. Most of the times problem to be solved is too complex to be solve in a single step. Than the problem is decomposed in to number of components in order to achieve an accurate cost estimate.

There are two approaches in decomposition technique:

1. Problem based estimation
2. Process based estimation

Problem based estimation:

In Problem based estimation we are estimating the cost on the basis of complete problem (Software) It means that we are going to define the cost of the software is totally depended on the Problem or software.

Lines of code and function points were described as measures from which productivity metrics can be computed.

LOC and FP data are used in two ways during software project estimation:

- (1) as an estimation variable to "size" each element of the software and
- (2) as baseline metrics collected from past projects and used in conjunction with estimation variables to develop cost and effort projections.

LOC and FP estimation are distinct estimation techniques. Yet both have a number of characteristics in common. The project planner begins with a bounded statement of software scope and from this statement attempts to decompose software into problem functions that can each be estimated individually. LOC or FP (the estimation variable) is then estimated for each function. Alternatively, the planner may choose another component for sizing such as classes or objects, changes, or business processes affected.

Baseline productivity metrics (e.g., LOC/pm or FP/pm) are then applied to the appropriate estimation variable, and cost or effort for the function is derived. Function estimates are combined to produce an overall estimate for the entire project.

It is important to note, however, that there is often substantial scatter in productivity metrics for an

organization, making the use of a single baseline productivity metric suspect. In general, LOC/pm or FP/pm averages should be computed by project domain. That is, projects should be grouped by team size, application area, complexity, and other relevant parameters. Local domain averages should then be

computed. When a new project is estimated, it should first be allocated to a domain, and then the appropriate domain average for productivity should be used in generating the estimate.

The LOC and FP estimation techniques differ in the level of detail required for decomposition and the target of the partitioning. When LOC is used as the estimation variable, decomposition is absolutely essential and is often taken to considerable levels of detail.

Estimation: The process approximating a value that can be used even if the data may be incomplete or unstable is referred to as estimation.

Problem based estimation:

- Begins with a statement of scope.
- The software is decomposed into problem functions.
- Estimating FP or LOC.
- Combine those estimates and produce an overall estimate.

Process based estimation:

- The functions of the software are identified.
- The framework is formulated.
- Estimate effort to complete each software function.
- Apply average labor rates, compute the total cost and compare the estimates.

Process based estimation:

The most common technique for estimating a project is to base the estimate on the process that will be used. That is, the process is decomposed into a relatively small set of tasks and the effort required to accomplish each task is estimated.

Like the problem-based techniques, process-based estimation begins with a definition of software functions obtained from the project scope. A series of software process activities must be performed for each function. Functions and related software process activities may be represented as part of a table.

Once problem functions and process activities are melded, the planner estimates the effort (e.g., person-months) that will be required to accomplish each software process activity for each software function. These data constitute the central matrix of the table. Average labor rates (i.e., cost/unit

effort) are then applied to the effort estimated for each process activity. It is very likely the labor rate will vary for each task. Senior staff heavily involved in early activities are generally more expensive than junior staff involved in later design tasks, code generation, and early testing.

Costs and effort for each function and software process activity are computed as the last step. If process-based estimation is performed independently of LOC or FP estimation, we now have two or three estimates for cost and effort that may be compared and reconciled. If both sets of estimates show reasonable agreement, there is good reason to believe that the estimates are reliable. If, on the other hand, the results of these decomposition techniques show little agreement, further investigation and analysis must be conducted.

Process based estimation:

- The functions of the software are identified.
- The framework is formulated.
- Estimate effort to complete each software function.
- Apply average labor rates, compute the total cost and compare the estimates.

Empirical estimation models:

Empirical (Realistic, Practical) estimation is a technique or model in which empirically derived formulas are used for predicting the data that are a required and essential part of the software project planning step.

These techniques are usually based on the data that is collected previously from a project and also based on some guesses, prior experience with the development of similar types of projects, and assumptions. It uses the size of the software to estimate the effort.

In this technique, refined guess of project parameters is made. Hence, these models are based on common sense. However, as there are many activities involved in empirical estimation techniques, this technique is formalized. For example Delphi technique and Expert Judgments technique.

The COCOMO Model.

Cocomo (Constructive Cost Model) is a regression model based on LOC, i.e number of Lines of Code. It is a procedural cost estimate model for software projects and often used as a process of reliably predicting the various parameters associated with making a project such as size, effort, cost, time and quality.

It was proposed by Barry Boehm in 1970 and is based on the study of 63 projects, which make it one of the best-documented models.

The key parameters which define the quality of any software products, which are also an outcome of the Cocomo are primarily Effort & Schedule:

1. Effort: Amount of labor that will be required to complete a task. It is measured in person-months units.
2. Schedule: Simply means the amount of time required for the completion of the job, which is, of course, proportional to the effort put. It is measured in the units of time such as weeks, months.

Different models of Cocomo have been proposed to predict the cost estimation at different levels, based on the amount of accuracy and correctness required. All of these models can be applied to a variety of projects, whose characteristics determine the value of constant to be used in subsequent calculations. These characteristics pertaining to different system types are mentioned below.

Boehm's definition of organic, semidetached, and embedded systems:

Organic – A software project is said to be an organic type if the team size required is adequately small, the problem is well understood and has been solved in the past and also the team members have a nominal experience regarding the problem.

Semi-detached – A software project is said to be a Semi-detached type if the vital characteristics such as team-size, experience, knowledge of the various programming environment lie in between that of organic and Embedded. The projects classified as Semi-Detached are comparatively less familiar and difficult to develop compared to the organic ones and require more experience and better guidance and creativity. Eg: Compilers or different Embedded Systems can be considered of Semi-Detached type.

Embedded – A software project with requiring the highest level of complexity, creativity, and experience requirement fall under this category. Such software requires a larger team size than the other two models and also the developers need to be sufficiently experienced and creative to develop such complex models.

All the above system types utilize different values of the constants used in Effort Calculations.

Types of Models: COCOMO consists of a hierarchy of three increasingly detailed and accurate forms. Any of the three forms can be adopted according to our requirements. These are types of COCOMO model:

1. Basic COCOMO Model
2. Intermediate COCOMO Model
3. Detailed COCOMO Model

The first level, Basic COCOMO can be used for quick and slightly rough calculations of Software Costs. Its accuracy is somewhat restricted due to the absence of sufficient factor considerations.

Intermediate COCOMO takes these Cost Drivers into account.

Detailed COCOMO additionally accounts for the influence of individual project phases, i.e in case of Detailed it accounts for both these cost drivers and also calculations are performed phase wise henceforth producing a more accurate result. These two models are further discussed below.

UNIT III: RISK MANAGEMENT

5. Risk Analysis And Management:

Software risks:

Software development is a multi stage approach of design, documentation, programming, prototyping, testing etc which follows a Software Development Life Cycle (SDLC) process. Different tasks are performed based on SDLC framework during software development. Developing and Maintaining software project involves risk in each step.

Most enterprises rely on software and ignoring the risks associated with any phase needs to be identified and managed/solved otherwise it creates unforeseen challenges for business.

Before analyzing different risks involved in software development, Let's first understand what is actually risk and why risk management is important for a business.

Risk and importance of risk management:

Risk is uncertain events associated with future events which have a probability of occurrence but it may or may not occur and if occurs it brings loss to the project. Risk identification and management are very important task during software project development because success and failure of any software project depends on it.

Various Kinds of Risks in Software Development:

Schedule Risk:

Schedule related risks refers to time related risks or project delivery related planning risks. The wrong schedule affects the project development and delivery. These risks are mainly indicates to running behind time as a result project development doesn't progress timely and it directly impacts to delivery of project. Finally if schedule risks are not managed properly it gives rise to project failure and at last it affect to organization/company economy very badly.

Some reasons for Schedule risks –

- Time is not estimated perfectly
- Improper resource allocation
- Tracking of resources like system, skill, staff etc
- Frequent project scope expansion
- Failure in function identification and its' completion

2. Budget Risk :

Budget related risks refers to the monetary risks mainly it occurs due to budget overruns. Always the financial aspect for the project should be managed as per decided but if financial aspect of project mismanaged then there budget concerns will arise by giving rise to budget risks. So proper finance distribution and management are required for the success of project otherwise it may lead to project failure.

Some reasons for Budget risks –

- Wrong/Improper budget estimation
- Unexpected Project Scope expansion
- Mismanagement in budget handling
- Cost overruns
- Improper tracking of Budget

3. Operational Risks :

Operational risk refers to the procedural risks means these are the risks which happen in day-to-day operational activities during project development due to improper process implementation or some external operational risks.

Some reasons for Operational risks –

- Insufficient resources
- Conflict between tasks and employees

- Improper management of tasks
- No proper planning about project
- Less number of skilled people
- Lack of communication and cooperation
- Lack of clarity in roles and responsibilities
- Insufficient training

4. **Technical Risks :**

Technical risks refers to the functional risk or performance risk which means this technical risk mainly associated with functionality of product or performance part of the software product.

Some reasons for Technical risks –

- Frequent changes in requirement
- Less use of future technologies
- Less number of skilled employee
- High complexity in implementation
- Improper integration of modules

5. **Programmatic Risks :**

Programmatic risks refers to the external risk or other unavoidable risks. These are the external risks which are unavoidable in nature. These risks come from outside and it is out of control of programs.

Some reasons for Programmatic risks –

- Rapid development of market
- Running out of fund / Limited fund for project development
- Changes in Government rules/policy
- Loss of contracts due to any reason

Risk is an expectation of loss, a potential problem that may or may not occur in the future. It is generally caused due to lack of information, control or time. A possibility of suffering from loss in software development process is called a software risk. Loss can be anything, increase in production cost, development of poor quality software, not being able to complete the project on time. Software risk exists because the future is uncertain and there are many known and unknown things that cannot be incorporated in the project plan. A software risk can be of two types (a) internal risks that are within the control of the project manager and (2) external risks that are beyond the control of project manager. Risk management is carried out to:

1. Identify the risk
2. Reduce the impact of risk
3. Reduce the probability or likelihood of risk
4. Risk monitoring

A project manager has to deal with risks arising from three possible cases:

1. Known knowns are software risks that are actually facts known to the team as well as to the entire project. For example not having enough number of developers can delay the project delivery. Such risks are described and included in the Project Management Plan.
2. Known unknowns are risks that the project team is aware of but it is unknown that such risk exists in the project or not. For example if the communication with the client is not of good level then it is not possible to capture the requirement properly. This is a fact known to the project team however whether the client has communicated all the information properly or not is unknown to the project.
3. Unknown Unknowns are those kind of risks about which the organization has no idea. Such risks are generally related to technology such as working with technologies or tools that you have no idea about because your client wants you to work that way suddenly exposes you to absolutely unknown unknown risks.

Software risk management is all about risk quantification of risk. This includes:

1. Giving a precise description of risk event that can occur in the project
2. Defining risk probability that would explain what are the chances for that risk to occur
3. Defining How much loss a particular risk can cause
4. Defining the liability potential of risk

Risk Management comprises of following processes:

1. Software Risk Identification
2. Software Risk Analysis
3. Software Risk Planning
4. Software Risk Monitoring

These Processes are defined below.

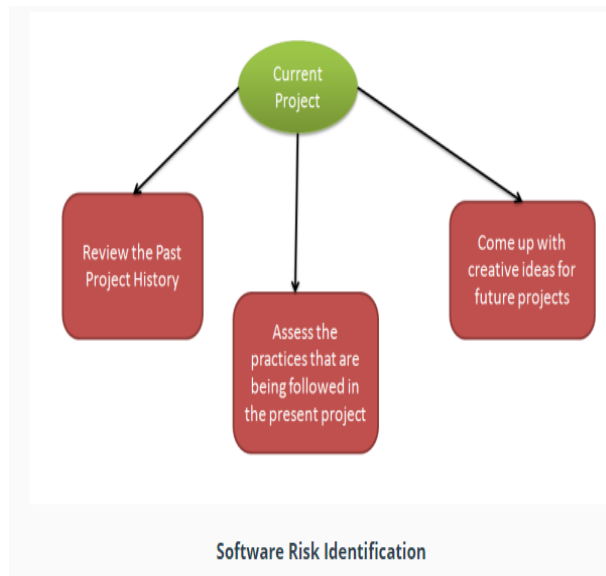
Software risk encompasses the probability of occurrence for uncertain events and their potential for loss within an organization. Risk management has become an important component of software development as organizations continue to implement more applications across a multiple technology, multi-tiered environment. Typically, software risk is viewed as a combination of robustness, performance efficiency, security and transactional risk propagated throughout the system.

Most organizations don't have a process to directly address the software risk that results from active custom software development. The traditional approach is to rely on testing – regression tests, system integration tests, performance tests and user integration tests. As you can see in the diagram, 30% of defects discovered in QA and live use are structural. And it is the structural defects that are the primary software risk exposure in the application lifecycle. Based on known software economics, that's 25 defects per function point that directly lead to software risk. Adding a structural quality gate to the QA process is imperative in order to measure and prevent software risk in mission critical systems. Most structural quality defects are actually not related to code quality issues, according to industry sources. It's a common misconception that code quality tools might address software risk. In reality, structural quality requires system level analysis in order to detect defects that pose software risk.

Software Risk Identification:

Risk identification is the process of determining risks that could potentially prevent the program, enterprise, or investment from achieving its objectives. It includes documenting and communicating the concern.

In order to identify the risks that your project may be subjected to, it is important to first study the problems faced by previous projects. Study the project plan properly and check for all the possible areas that are vulnerable to some or the other type of risks. The best ways of analyzing a project plan is by converting it to a flowchart and examine all essential areas. It is important to conduct few brainstorming sessions to identify the known unknowns that can affect the project. Any decision taken related to technical, operational, political, legal, social, internal or external factors should be evaluated properly.



In this phase of Risk management you have to define processes that are important for risk identification. All the details of the risk such as unique Id, date on which it was identified, description and so on should be clearly mentioned.

Risk projection:

Risk projection, also called risk estimation, attempts to rate each risk in two ways—the likelihood or probability that the risk is real and the consequences of the problems associated with the risk, should it occur. The project planner, along with other managers and technical staff, performs four risk projection activities:

- (1) Establish a scale that reflects the perceived likelihood of a risk.
- (2) Delineate the consequences of the risk.
- (3) Estimate the impact of the risk on the project and the product.
- (4) Note the overall accuracy of the risk projection so that there will be no misunderstandings.

Developing a Risk Table

A risk table provides a project manager with a simple technique for risk projection. A project team begins by listing all risks (no matter how remote) in the first column of the table. This can be accomplished with the help of the risk item checklists. Each risk is categorized in the second column

(e.g., PS implies a project size risk, BU implies a business risk). The probability of occurrence of each risk is entered in the next column of the table. The probability value for each risk can be estimated by team members individually. Individual team members are polled in round-robin fashion until their assessment of risk probability begins to converge.

Risk refinement:

Process of restating the risks as a set of more detailed risks that will be easier to mitigate, monitor, and manage.

CTC (condition-transition-consequence) format may be a good representation for the detailed risks (e.g. given that <condition> then there is a concern that (possibly) <consequence>).

During early stages of project planning, a risk may be stated quite generally. As time passes and more is learned about the project and the risk, it may be possible to refine the risk into a set of more detailed risks, each somewhat easier to mitigate, monitor, and manage.

One way to do this is to represent the risk in condition-transition-consequence (CTC) format . That is, the risk is stated in the following form: Given that <condition> then there is concern that (possibly) <consequence>.

Using the CTC format for the reuse risk noted in Section 6.4.2, we can write:

Given that all reusable software components must conform to specific design standards and that some do not conform, then there is concern that (possibly) only 70 percent of the planned reusable modules may actually be integrated into the as-built system, resulting in the need to custom engineer the remaining 30 percent of components.

RISK REFINEMENT

- *During early stages of project planning, a risk may be stated quite generally. As time passes and more is learned about risk, it may be possible to refine the risk into a set of more detailed risks each somewhere easy to monitor and manage.*
- *One way to do this is to represent the risk in Condition Transition Consequence format:
Given that<condition>then there is concern that (possibly)<consequence>.*
- *Refinement helps to isolate the underlying risks and lead to easier analysis and response.*

Risk Refinement

- Also called Risk assessment
- Refines the risk table in reviewing the risk impact based on the following three factors
 - a. Nature: Likely problems if risk occurs
 - b. Scope: Just how serious is it?
 - c. Timing: When and how long

Risk mitigation (justification):

Risk mitigation can be defined as taking steps to reduce adverse effects. There are four types of risk mitigation strategies that hold unique to Business Continuity and Disaster Recovery. When mitigating risk, it's important to develop a strategy that closely relates to and matches your company's profile.



4. Risk Acceptance

Risk acceptance does not reduce any effects however it is still considered a strategy. This strategy is a common option when the cost of other risk management options such as avoidance or limitation may outweigh the cost of the risk itself. A company that doesn't want to spend a lot of money on avoiding risks that do not have a high possibility of occurring will use the risk acceptance strategy.

5. Risk Avoidance

Risk avoidance is the opposite of risk acceptance. It is the action that avoids any exposure to the risk whatsoever. It's important to note that risk avoidance is usually the most expensive of all risk mitigation options.

6. Risk Limitation

Risk limitation is the most common risk management strategy used by businesses. This strategy limits a company's exposure by taking some action. It is a strategy employing a bit of risk acceptance along with a bit of risk avoidance or an average of both. An example of risk limitation would be a company accepting that a disk drive may fail and avoiding a long period of failure by having backups.

7. Risk Transference

Risk transference is the involvement of handing risk off to a willing third party. For example, numerous companies outsource certain operations such as customer service, payroll services, etc. This can be beneficial for a company if a transferred risk is not a core competency of that company. It can also be used so a company can focus more on its core competencies.

Monitoring and management of Risks.

Risk monitoring is the ongoing process of managing risk. Risk management often has an initial phase that involves identifying risk, agreeing to treatments and designing controls. Risk monitoring is the process of tracking risk management execution and continuing to identify and manage new risks.

Risk Monitoring

Risk monitoring activities implement the risk monitoring strategy by gathering information through automated or manual means, alerting or reporting on information relevant to intended purposes for risk monitoring, and providing inputs to ongoing risk assessment and response processes. Depending on risk assumptions, constraints, priorities, and tolerance levels, the set of risk monitoring practices actually implemented at any one time may differ from what is documented in the risk monitoring strategy. NIST guidance recommends that organizations consider risk monitoring from an organizational perspective and coordinate monitoring practices across all three tiers to support the achievement of overall risk management goals and avoid potential duplication of implemented monitoring activities.

The purpose of risk monitoring is to address how risk will be monitored. This includes verifying compliance with the risk response decisions by ensuring that the organization implements the risk response measures (and any information security requirements), determines the ongoing effectiveness of risk response measures, and identifies any changes that would impact the risk posture. Risk monitoring activities at the various levels of the organization (or with other organizational entities) should be coordinated and communicated. This can include sharing risk assessment results that would have an organization-wide impact to risk responses being planned or

implemented. The organization should also consider the tools and technologies that will be needed to facilitate monitoring and the frequency necessary for effectively monitoring risks, including the changes that would impact responses to risks.

Management of Risks.

It have concern with managing all types of risks as per their seriousness it means there are three categories of risk.

1. Most dangerous risks.
2. Dangerous risks.
3. Normal risks.

In this it is going to focus on the most denatures risk.

6. Software Quality Assurance:

Basic concepts:

Software quality assurance (SQA):

Software quality assurance (SQA) is a process which assures that all software engineering processes, methods, activities and work items are monitored and comply against the defined standards.

These defined standards could be one or a combination of any like ISO 9000, CMMI model, ISO15504, etc.

Quality Assurance in Software Testing is defined as a procedure to ensure the quality of software products or services provided to the customers by an organization.

Quality assurance focuses on improving the software development process and making it efficient and effective as per the quality standards defined for software products. Quality Assurance is popularly known as QA Testing.

Software Quality assurance makes sure the project will be completed based on the previously agreed specifications, standards and functionality required without defects and possible problems. It monitors and tries to improve the development process from the beginning of the project to ensure this. It is oriented to “prevention”.

Software QA is involved in the project from the beginning. This helps the teams communicate and understand the problems and concerns, also gives time to set up the testing environment and configuration. On the other hand, actual testing starts after the test plans are written, reviewed and approved based on the design documentation.

Software quality assurance testing process identifies problems early on in the development process, isolating project risk and maximizing ROI for clients. Verification is preventing mechanism to detect possible failures before the testing begin. It involves reviews, meetings, evaluating documents, plans,

code, inspections, specifications etc. Validation occurs after verification and it's the actual testing to find defects against the functionality or the specifications.

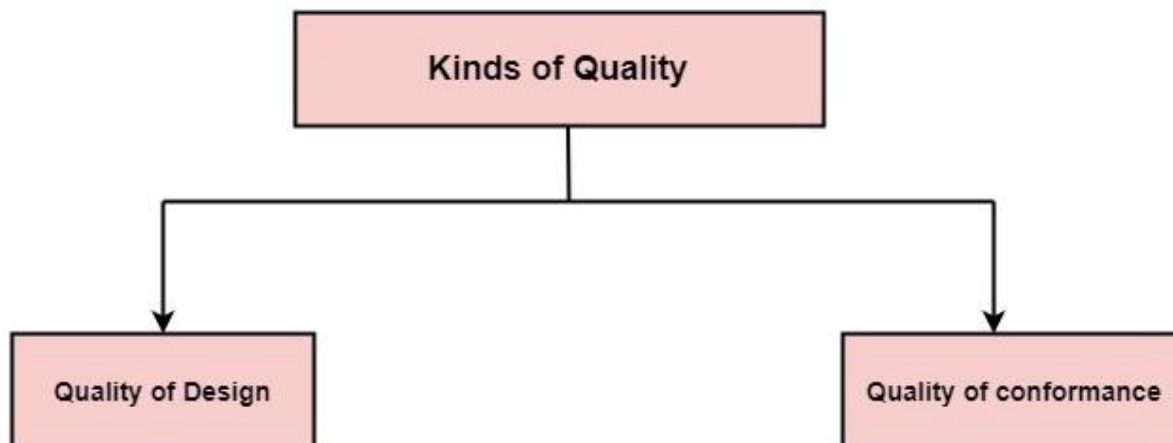
This can include processes such as requirements gathering and documentation, source code control, code review, change management, configuration management, release management and the actual testing of the software.

Software QA is often confused with Software Testing, but should not be. These QA methods, in manufacturing, proved themselves to work (in Sales, Customer satisfaction and the right cost of production, i.e. PROFIT) and were adopted all over the world. QA groups (for manufacturing) became the norm.

Quality:

Quality defines to any measurable characteristics such as correctness, maintainability, portability, testability, usability, reliability, efficiency, integrity, reusability, and interoperability.

There are two kinds of Quality:



Quality of Design: Quality of Design refers to the characteristics that designers specify for an item.

The grade of materials, tolerances, and performance specifications that all contribute to the quality of design.

Quality of conformance: Quality of conformance is the degree to which the design specifications are followed during manufacturing.

Greater the degree of conformance, the higher is the level of quality of conformance.

Quality control:

Quality Control: Quality Control involves a series of inspections, reviews, and tests used throughout the software process to ensure each work product meets the requirements placed upon it.

Quality control includes a feedback loop to the process that created the work product.

Quality control (QC) is a process through which a business seeks to ensure that product **quality** is maintained or improved.

Quality control involves testing of units and determining if they are within the specifications for the final product.

Quality assurance:

Software Quality Assurance (SQA) is a set of activities for ensuring quality in software engineering processes.

It ensures that developed software meets and complies with the defined or standardized quality specifications.

SQA is an ongoing process within the Software Development Life Cycle (SDLC) that routinely checks the developed software to ensure it meets the desired quality measures.

SQA practices are implemented in most types of software development, regardless of the underlying software development model being used.

SQA incorporates and implements software testing methodologies to test the software. Rather than checking for quality after completion, SQA processes test for quality in each phase of development, until the software is complete.

With SQA, the software development process moves into the next phase only once the current/previous phase complies with the required quality standards. SQA generally works on one or more industry standards that help in building software quality guidelines and implementation strategies.

It includes the following activities –

- Process definition and implementation
- Auditing
- Training

Processes could be –

- Software Development Methodology
- Project Management
- Configuration Management
- Requirements Development/Management
- Estimation
- Software Design
- Testing, etc.

Once the processes have been defined and implemented, Quality Assurance has the following responsibilities –

- Identify the weaknesses in the processes
- Correct those weaknesses to continually improve the process

Quality assurance (QA) is a way of preventing mistakes and defects in manufactured products and avoiding problems when delivering products or services to customers; which ISO 9000 defines as "part of **quality** management focused on providing confidence that **quality** requirements will be fulfilled".

Quality assurance encompasses the processes and procedures that systematically monitor different aspects of a service or facility. Through audits and other forms of assessment, quality assurance efforts detect and correct problems or variances that fall outside established standards or requirements.

The term "quality assurance" is sometimes used interchangeably with "quality control," another facet of the management process. However, quality control pertains to the actual fulfillment of whatever quality requirements have been put in place. Quality assurance is checking in on quality control methods to ensure they're working as planned.

Most businesses utilize some form of quality assurance in production, from manufacturers of consumer packaged goods to software development companies. Some companies may even establish a quality assurance department with employees that focus solely on quality assurance.

Cost of quality:

Cost of quality is a method for calculating the costs companies incur ensuring that products meet quality standards, as well as the costs of producing goods that fail to meet quality standards.

The goal of calculating cost of quality is to create an understanding of how quality impacts the bottom line.

Whether it's the cost of scrap and rework associated with poor quality, or the expense of audits and maintenance associated with good quality, both count. Cost of quality gives manufacturers an opportunity to analyze, and thus improve their quality operations.

COST OF QUALITY (COQ) is a measure that quantifies the cost of control/conformance and the cost of failure of control/non-conformance. In other words, it sums up the costs related to prevention and detection of defects and the costs due to occurrences of defects.

Definition by ISTQB: cost of quality: The total costs incurred on quality activities and issues and often split into prevention costs, appraisal costs, internal failure costs and external failure costs.

Definition by QAI: Money spent beyond expected production costs (labor, materials, equipment) to ensure that the product the customer receives is a quality (defect free) product. The Cost of Quality includes prevention, appraisal, and correction or repair costs.

Software quality assurance(SQA):

Software quality assurance (SQA) is a process which assures that all software engineering processes, methods, activities and work items are monitored and comply against the defined standards. These defined standards could be one or a combination of any like ISO 9000, CMMI model, ISO15504, etc.

SQA incorporates all software development processes starting from defining requirements to coding until release. Its prime goal is to ensure quality.

Software quality assurance (SQA) is a means and practice of monitoring the software engineering processes and methods used in a project to ensure proper quality of the software. It may include ensuring conformance to standards or models, such as ISO/IEC 9126 (now superseded by ISO 25010), SPICE or CMMI.

It includes standards and procedures that managers, administrators or even developers may use to review and audit software products and activities to verify that the software meets quality criteria which link to standards. According to Automotive SPICE (which is based on ISO/IEC 15504), software quality assurance is a supporting process (SUP.1) that provides the independent assurance that all work products, activities and processes comply with the predefined plans and quality strategies.

SQA encompasses the entire software development process, including requirements engineering, software design, coding, code reviews, source code control, software configuration management, testing, release management and software integration. It is organized into goals, commitments, abilities, activities, measurements, verification and validation.

SQA involves a three-prong approach:

1. Organization-wide policies, procedures and standards
2. Project-specific policies, procedures and standards
3. Compliance to appropriate procedures

Software Quality Assurance (SQA) is a set of activities for ensuring quality in software engineering processes. It ensures that developed software meets and complies with the defined or standardized quality specifications. SQA is an ongoing process within the Software Development Life Cycle (SDLC) that routinely checks the developed software to ensure it meets the desired quality measures.

SQA practices are implemented in most types of software development, regardless of the underlying software development model being used. SQA incorporates and implements software testing methodologies to test the software. Rather than checking for quality after completion, SQA processes test for quality in each phase of development, until the software is complete. With SQA, the software development process moves into the next phase only once the current/previous phase complies with the required quality standards. SQA generally works on one or more industry standards that help in building software quality guidelines and implementation strategies.

It includes the following activities –

- Process definition and implementation
- Auditing
- Training

Processes could be –

- Software Development Methodology
- Project Management
- Configuration Management

- Requirements Development/Management
- Estimation
- Software Design
- Testing, etc.

Formal technical review: (FTR)

Formal Technical Review (FTR) is a software quality control activity performed by software engineers.

Objectives of formal technical review (FTR):

Some of these are:

- 1. Useful to uncover error in logic, function and implementation for any representation of the software.**
- 2. The purpose of FTR is to verify that the software meets specified requirements.**
- 3. To ensure that software is represented according to predefined standards.**
- 4. It helps to review the uniformity in software that is development in a uniform manner.**
- 5. To makes the project more manageable.**

In addition, the purpose of FTR is to enable junior engineer to observer the analysis, design, coding and testing approach more closely.

FTR also works to promote back up and continuity become familiar with parts of software they might not have seen otherwise.

Actually, FTR is a class of reviews that include walkthroughs, inspections, round robin reviews and other small group technical assessments of software. Each FTR is conducted as meeting and is considered successfully only if it is properly planned, controlled and attended.

The review meeting:

Each review meeting should be held considering the following constraints-

Involvement of people:

Between 3, 4 and 5 people should be involve in the review.

Advance preparation should occur but it should be very short that is at the most 2 hours of work for every person.

The short duration of the review meeting should be less than two hour. Gives these constraints, it should be clear that an FTR focuses on specific (and small) part of the overall software.

At the end of the review, all attendees of FTR must decide what to do.

Accept the product without any modification.

Reject the project due to serious error (Once corrected, another app need to be reviewed), or

Accept the product provisional (minor errors are encountered and are should be corrected, but no additional review will be required).

The decision was made, with all FTR attendees completing a sign-of indicating their participation in the review and their agreement with the findings of the review team.

Review reporting and record keeping :-

During the FTR, the reviewer actively records all issues that have been raised.

At the end of the meeting all these issues raised are consolidated and a review list is prepared.

Finally, a formal technical review summary report is prepared.

It answers three questions :-

What was reviewed ?

Who reviewed it ?

What were the findings and conclusions ?

Review guidelines :-

Guidelines for the conducting of formal technical reviews should be established in advance. These

guidelines must be distributed to all reviewers, agreed upon, and then followed. A review that is unregistered can often be worse than a review that does not minimum set of guidelines for FTR.

Review the product, not the manufacture (producer).

Take written notes (record purpose)

Limit the number of participants and insists upon advance preparation.

Develop a checklist for each product that is likely to be reviewed.

Allocate resources and time schedule for FTRs in order to maintain time schedule.

Conduct meaningful training for all reviewers in order to make reviews effective.

Reviews earlier reviews which serve as the base for the current review being conducted.

Set an agenda and maintain it.

Separate the problem areas, but do not attempt to solve every problem notes.

Limit debate and rebuttal.

UNIT IV: TESTINGS

7. Coding and Unit Testing

Programming principles and guidelines:

Bad code comes in many forms.

Messy code, massive if-else chains, programs that break with one adjustment, variables that don't make sense.

The program might work once but will never hold up to any scrutiny.

If you want to be a programmer, **don't settle for shortcuts.**

Aim to write code that is easy to maintain. Easy for you to maintain, and easy for any other developer on your team to maintain.

How do you write effective code? You write good code by being disciplined with programming principles.

Here are 10 programming principles that will make you a better coder.

1. Keep It Simple, Stupid (KISS)

It sounds a little harsh, but it's a coding principle to live by. What does this mean?

It means you should be writing code as simple as possible. Don't get caught up in trying to be overly clever or showing off with a paragraph of advanced code. If you can write a script in one line, write it in one line.

Here's a simple function:

```
function addNumbers(num1,num2){  
  return num1 + num2;  
}
```

Pretty simple. It's easy to read and you know exactly what is going on.

Use clear variable names. Take advantage of coding libraries to use existing tools. Make it easy to come back after six months and get right back to work. Keeping it simple will save you the headache.

2. Write DRY Code

The Don't Repeat Yourself (DRY) principle means, plainly, not repeating code. It's a common coding mistake. When writing code, avoid duplication of data or logic. If you've ever copied and pasted code within your program, it's not DRY code.

Take a look at this script:

```
function addNumberSequence(number){  
  number = number + 1;  
  number = number + 2;  
  number = number + 3;  
  number = number + 4;  
  number = number + 5;  
  return number;  
}
```

Instead of duplicating lines, try to find an algorithm that uses iteration. For loops, and while loops are ways to control code that needs to run multiple times.

DRY code is easy to maintain. It's easier to debug one loop that handles 50 repetitions than 50 blocks of code that handle one repetition.

Instead of duplicating lines, try to find an algorithm that uses iteration. For loops, and while loops are ways to control code that needs to run multiple times.

DRY code is easy to maintain. It's easier to debug one loop that handles 50 repetitions than 50 blocks of code that handle one repetition.

3. Open/Closed

This principle means you should aim to make your code open to extension but closed to modification. This is an important principle when releasing a library or framework that others will use.

For example, suppose you're maintaining a GUI framework. You could release for coders to directly modify and integrate your released code. But what happens when you release a major update four months later?

Their code will break. This will make engineers unhappy. They won't want to use your library for much longer, no matter how helpful it may be.

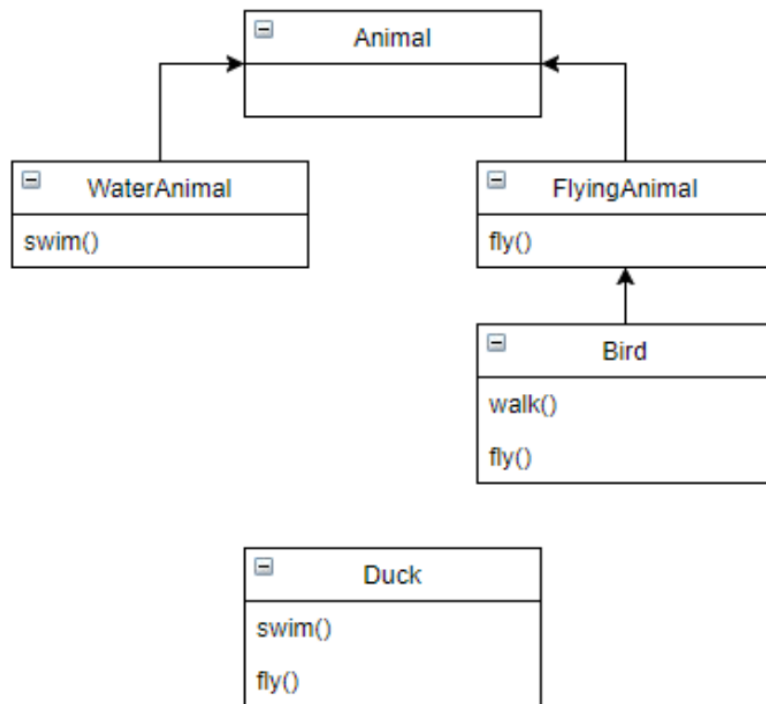
Instead, release code that prevents direct modification and encourages extension. This separates core behavior from modified behavior. The code is more stable and easier to maintain.

4. Composition (Arrangement, Alignment, Structure)over Inheritance.

If you write code using object-oriented programming you're going to find this useful. The composition over inheritance principle states: objects with complex behaviors should contain instances of objects with individual behaviors. They should not inherit a class and add new behaviors.

Relying on inheritance causes two major issues. First, the inheritance hierarchy can get messy in a hurry.

You also have less flexibility for defining special-case behaviors. Let's say you want to implement behaviors to share:



Composition programming is a lot cleaner to write, easier to maintain and allows flexibility defining behaviors. Each individual behavior is its own class. You can create complex behaviors by combining individual behaviors.

5. Single Responsibility

The single responsibility principle says that every class or module in a program should only provide one specific functionality.

As Robert C. Martin puts it, "A class should have only one reason to change."

Classes and modules often start off this way. Be careful not to add too many responsibilities as classes get more complicated. Refactor and break them up into smaller classes and modules.

The consequence of overloading classes is twofold. First, it complicates debugging when you're trying to isolate a certain module for troubleshooting. Second, it becomes more difficult to create additional functionality for a specific module.

6. Separation of Concerns

The separation of concerns principle is an abstract version of the single responsibility principle. This idea states that a program should be designed with different containers, and these containers should not have access to each other.

A well-known example of this is the model-view-controller (MVC) design. MVC separates a program into three distinct areas: the data (model), the logic (controller), and what the page displays (view). Variations of MVC are common in today's most popular web frameworks.

For example, the code that handles the database doesn't need to know how to render the data in the browser. The rendering code takes input from the user, but the logic code handles the processing. Each piece of code is completely independent.

The result is code that is easy to debug. If you ever need to rewrite the rendering code, you can do so without worrying about how the data gets saved or the logic gets processed.

7. You Aren't Going to Need It (YAGNI)

This principle means you should never code for functionality on the chance that you may need in the future. Don't try and solve a problem that doesn't exist.

In an effort to write DRY code, programmers can violate this principle. Often inexperienced programmers try to write the most abstract and generic code they can. Too much abstraction causes bloated code that is impossible to maintain.

Only apply the DRY principle only when you need to. If you notice chunks of code written over and over, then abstract them. Don't think too far out at the expense of your current code batch.

8. Document Your Code

Any senior developer will stress the importance of documenting your code with proper comments. All languages offer them and you should make it a habit to write them. Leave comments to explain objects, enhance variable definitions, and make functions easier to understand.

Here's a JavaScript function with comments guiding you through the code:

```
//This function will add 5 to the input if odd, or return the number if even
function evenOrOdd(number){
  //Determine if the number is even
  if(number % 2 == 0){
    return number;
  }
  //If the number is odd, this will add 5 and return
  else {
    return number + 5;
  }
}
```

Leaving comments is a little more work while you're coding, and you understand your code pretty well right?

Leave comments anyway!

Try writing a program, leaving it alone for six months, and come back to modify it. You'll be glad you documented your program instead of having to pour over every function to remember how it works. Work on a coding team? Don't frustrate your fellow developers by forcing them to decipher your syntax.

Overview

Programming principles have helped me over the years in becoming a better programmer, and I believe, this article will help any developer become more efficient and able to produce code which is

easier to maintain. By following these coding principles, you can save development and maintenance time, and conquer lots of other bottlenecks which generally arise in later development phases.

What are Programming principles?

Programming is the process of coding, testing, troubleshooting, debugging and maintaining a system. Programming principles help you to write excellent quality of code and maintain a good coding practice.

Why should a developer follow the principles?

When I started my adventure with programming in C language, I wrote complete spaghetti code. After that, I started coding in C#. Here also, I did the same thing. It means each file had many responsibilities, no abstraction, and it was difficult to debug and fix issues. I am not only one person working on applications; my team is working on the same. Later, I started thinking how to improve quality of the code. And the answer is Coding Principles.

Actually, writing programs is often a matter of personal taste but there are certain principles or guidelines that should be followed within your own style in order to make programs easy to maintain and understand by both, you and others, and also principles guide the creation of stable, scalable, and robust code.

Incrementally developing code:

See the pdf file for reaming all notes.