Introduction to software engineering

- 1.1 Introduction to software
- 1.2 Program Vs software
- 1.3 Software components
- 1.4 . Characteristics of software
- 1.5 Types of software
- 1.6 Generic view of software engineering
- 1.7 Software process and software process model

- 1.Software Engineering :The term is made of two words, software and engineering. Software is more than just a program code. A program is an executable code, which serves some computational purpose. Software is considered to be collection of executable programming code, associated libraries and documentations. Software, when made for a specific requirement is called software product.
- Engineering on the other hand, is all about developing products, using well-defined, scientific principles and methods.

- 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.

Software Engineering Body of Knowledge

- The Software Engineering Body of Knowledge (SWEBOK) is an international standard ISO/IEC TR 19759:2005[1] specifying a guide to the generally accepted Software Engineering Body of Knowledge.
- The Guide to the Software Engineering Body of Knowledge (SWEBOK Guide) has been created through cooperation among several professional bodies and members of industry and is published by the IEEE Computer Society (IEEE).

Programs v/s Software

- Software is a broad term that covers the programs and components that it required to run. Software consists the files, whereas a program can itself be a file. Along with these differences, there are various other comparisons between both terms.

On the basis of	Program	Software
Definition	A computer program is a set of instructions that is used as a process of creating a software program by using programming language.	Software is a set of programs that enables the hardware to perform a specific task.
Types	Programs do not have further categorization.	The software can be of three types: system software, application software, and programming software.
User Interface	A program does not have a user interface.	Every software has a user interface that may be in graphical format or in the form of a command prompt.

Size	Programs are smaller in size, and their size exists between Kilobyte (Kb) to a megabyte (Mb).	Software's are larger in size, and their size exists between megabytes (Mb) to gigabytes (Gb).
Time taken	A program takes less time to be developed.	Whereas software requires more time to be developed.
Features and functionality	A program includes fewer features and limited functionalities.	It has more features and functionalities.
Development approach	The development approach of a program is unorganized, unplanned, and unprocedural.	The development approach of software is well planned, organized, and systematic.
Documentation	There is a lack of documentation in the program.	Softwares are properly documented.
Examples	Examples of the program are - video games, malware, and many more.	Examples of software are - Adobe Photoshop, Adobe Reader, Google Chrome, and

SOFTWARE CHARACTERISTICS

- **Software is developed :** It is not manufactured. It is not something that will automatically roll out of an assembly line. It ultimately depend on individual skill and creative ability
- **Software does not Wear Out :** Software is not susceptible to the environmental melodies and it does not suffer from any effects with time

- **Software is Highly Malleable** : In case of software one can modify the product itself rather easily without necessary changes.

- **Most Software is Created and Not Assembled from Existing Components**

- **Functionality:**
  It refers to the suitability, accuracy, interoperability, compliance, security of software which is measured as degree of performance of the software against its intended purpose.

- **Reliability:**
  Refers to the recoverability, fault tolerance, maturity of software, which is basically a capability of the software that provide required functionality under the given situations.

- **Efficiency:**
  It is the ability of the software to use resources of system in the most effective and efficient manner. Software must make effective use of syetm storage and execute command as per required timing.

- **Usability:**
  It is the extent to which the software can be utilized with ease and the amount of effort or time required to learn how to use the software.
- **Maintainability:**
  It is the ease with which the modifications can be made in a software to extend or enhance its functionality, improve its performance, or resolve bugs.
- **Portability:**
  It is the ease with which software developers can relaunch software from one platform to another, without (or with minimum) changes. In simple terms, software must be made in way that it should be platform independent.

SOFTWARE COMPONENTS

- Off the shelf Components : Existing software that can be acquired from a third party.
- Full Experience Components : Existing past projects that are similar to the software to be built for the current project and team members have full experience.
- Partial Experience components : Existing past project that are related to the software to be built for current project but needs substantial modifications
- New Components : Software components that must be built by the software team specifically for the needs of the current project

Software Process

- A software process (also knows as software methodology) is a set of related activities that leads to the production of the software.These activities may involve the development of the software from the scratch, or, modifying an existing system.
- Any software process must include the following four activities:
- 1. Software specification (or requirements engineering): Define the main functionalities of the software and the constrains around them.
- 2. Software design and implementation: The software is to be designed and programmed.
- 3. Software verification and validation:The software must conforms to it's specification and meets the customer needs.
- 4. Software evolution (software maintenance):The software is being modified to meet customer and market requirements changes.

Software Process Framework:

- A process framework establishes the foundation for a complete software process by identifying a small number of framework activities that are applicable to all software projects, regardless of size or complexity. It also includes a set of umbrella activities that are applicable across the entire software process.
- Some most applicable framework activities are described below.

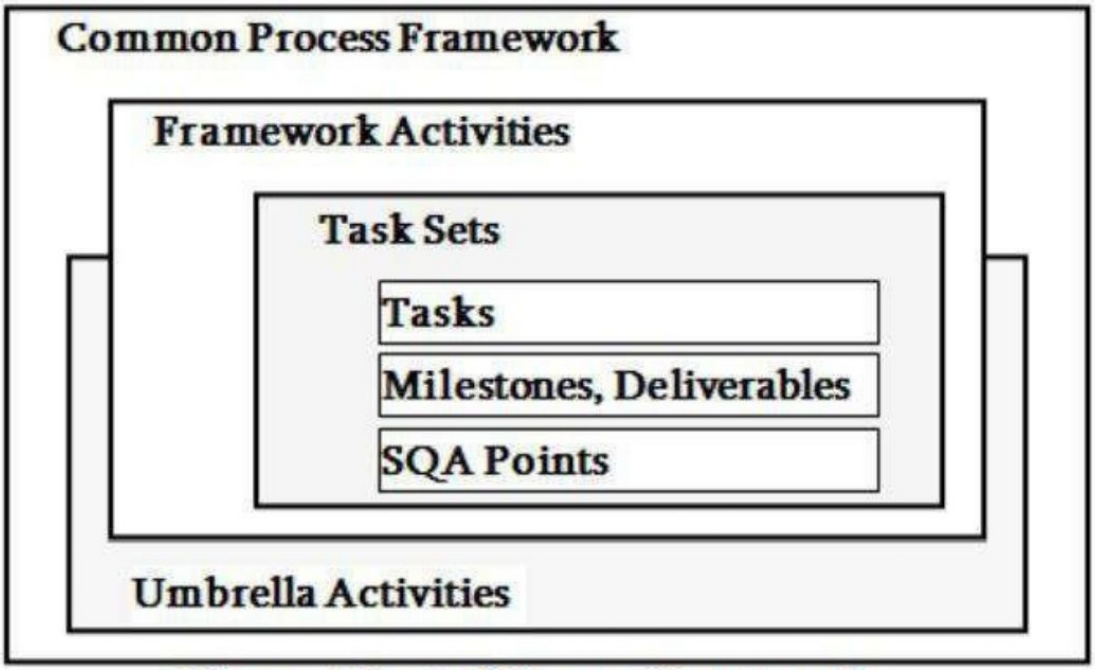


Figure: Chart of Process Framework

Elements of software process:

- They are different elements of software process.
- 1. Communication:This activity involves heavy communication with customers and other stakeholders in order to gather requirements and other related activities.
- 2. Planning: Here a plan to be followed will be created which will describe the technical tasks to be conducted, risks, required resources, work schedule etc.
- 3. Modeling: A model will be created to better understand the requirements and design to achieve these requirements.
- 4. Construction: Here the code will be generated and tested.
- 5.Deployment: Here, a complete or partially complete version of the software is represented to the customers to evaluate and they give feedbacks based on the evaluation.

IMPORTANCE OF SOFTWARE ENGINEERING

- **1. Reduces complexity**

  Big software are always complex and difficult to develop. Software engineering has a great solution to decrease the complexity of any project..

- **2. To minimize software cost**

  Software requires a lot of hard work and software engineers are highly paid professionals. But in software engineering, programmers plan everything and reduce all those things that are not required. In turn, cost for software productions becomes less.

- **3. To decrease time**

  If you are making big software then you may need to run many code to get the ultimate running code. This is a very time consuming So if you are making your software according to software engineering approach then it will reduce a lot of time.

- **4. Handling big projects**

  Big projects are not made in few days and they require lots of patience, So to handle big projects without any problem, organization has to go for software engineering approach.

- **5. Reliable software**

  Software should be reliable, means if you have delivered the software then it should work for at least it's given time

- **6. Effeteness**

  Effectiveness comes if anything has made according to the standards. So Software becomes more effective in performance with the help of software engineering.

- **7. Productivity**

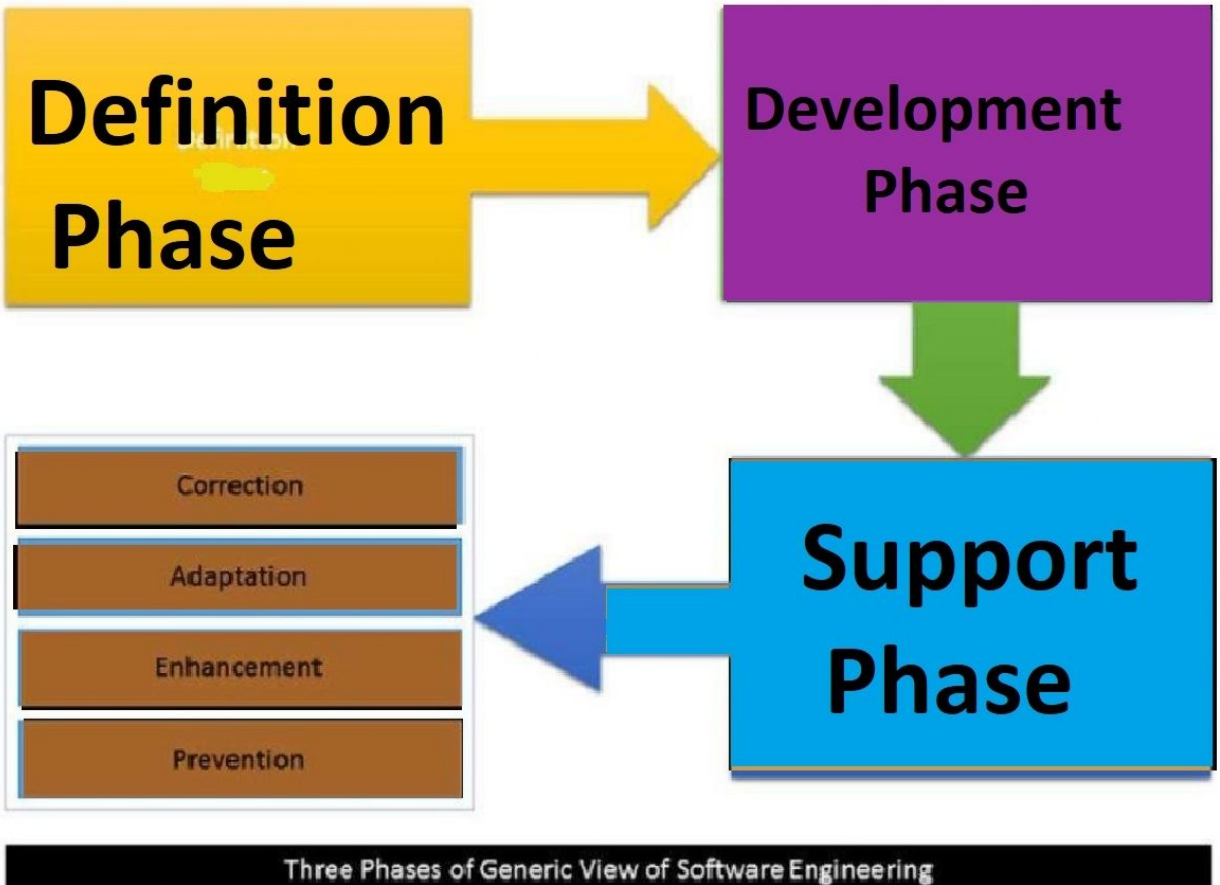
  If programs fails to meet its standard at any stage, then programmers always improves the code of software to make it sure that software maintains its standards.

Changing Nature of Software :

- The nature of software has changed a lot over the years.
- **1.System software:** Infrastructure software come under this category like compilers, operating systems, editors, drivers, etc. Basically system software is a collection of programs to provide service to other programs.
- **2. Real time software:** These software are used to monitor, control and analyze real world events as they occur. An example may be software required for weather forecasting. Such software will gather and process the status of temperature, humidity and other environmental parameters to forcast the weather.
- **3. Embedded software:** This type of software is placed in "Read-Only- Memory (ROM)"of the product and control the various functions of the product.The product could be an aircraft, automobile, security system, signalling system, control unit of power plants, etc.The embedded software handles hardware components and is also termed as intelligent software
- **4. Business software :** This is the largest application area. The software designed to process business applications is called business software. Business software could be payroll, file monitoring system, employee management, account management. It may also be a data warehousing tool which helps us to take decisions based on available data. Management information system, enterprise resource planning (ERP) and such other software are popular examples of business software.

- **5. Personal computer software :** The software used in personal computers are covered in this category. Examples are word processors, computer graphics, multimedia and animating tools, database management, computer games etc. This is a very upcoming area and many big organisations are concentrating their effort here due to large customer base.
- **6. Artificial intelligence software:** Artificial Intelligence software makes use of non numerical algorithms to solve complex problems that are not amenable to computation or straight forward analysis. Examples are expert systems, artificial neural network,signal processing software etc
- **7.Web based software:** The software related to web applications come under this category. Examples are HTML, Java, Perl, DHTML etc

A Generic View of Software Engineering



Three Phases of Generic View of Software Engineering

- **1. Definition Phase:**
- The definition phase focuses on "what". That is, during definition, the software engineer attempts to identify what information is to be processed, what function and performance are desired, what system behavior can be expected, what interfaces are to be established, what design constraints exist, and what validation criteria are required to define a successful system. During this, three major tasks will occur in some form: system or information engineering, software project planning and requirements analysis.

- **2. Development Phase:**
- The development phase focuses on "how".That is, during development a software engineer attempts to define how data are to be structured, how function is to be implemented within a software architecture, how interfaces are to be characterized, how the design will be translated into a programming language, and how testing will be performed. During this, three specific technical tasks should always occur; software design, code generation, and software testing.

- **3. Support Phase:**
- The support phase focuses on "change" associated with error correction, adaptations required as the software's environment evolves, and changes due to enhancements brought about by changing customer requirements. Four types of change are encountered during the support phase:

- **Correction.** Even with the best quality assurance activities, it is likely that the customer will uncover defects in the software. Corrective maintenance changes the software to correct defects.

- **Adaptation.** Over time, the original environment (e.g., CPU, operating system, business rules, external product characteristics) for which the software was developed is likely to change. Adaptive maintenance results in modification to the software to accommodate changes to its external environment.

- **Enhancement.** As software is used, the customer/user will recognize additional functions that will provide benefit. Perfective maintenance extends the software beyond its original functional requirements.

- **Prevention.** Computer software deteriorates due to change, and because of this, preventive maintenance, often called software reengineering, must be conducted to enable the software to serve the needs of its end users. In essence, preventive maintenance makes changes to computer programs so that they can be more easily corrected, adapted, and enhanced.

Software Processes

- The term **software** specifies to the set of computer programs, procedures and associated documents (Flowcharts, manuals, etc.) that describe the program and how they are to be used.
- A software process is the set of activities and associated outcome that produce a software product. Software engineers mostly carry out these activities.These are four key process activities, which are common to all software processes. These activities are:
- **Software specifications:** The functionality of the software and constraints on its operation must be defined.
- **Software development:** The software to meet the requirement must be produced.
- **Software validation:** The software must be validated to ensure that it does what the customer wants.
- **Software evolution:** The software must evolve to meet changing client needs.

- The Software Process Model

- A software process model is a specified definition of a software process, which is presented from a particular perspective. Models, by their nature, are a simplification, so a software process model is an abstraction of the actual process, which is being described.
- Some examples of the types of software process models that may be produced are:

  A workflow model: This shows the series of activities in the process along with their inputs, outputs and dependencies. The activities in this model perform human actions.

- **2. A dataflow or activity model:** This represents the process as a set of activities, each of which carries out some data transformations. It shows how the input to the process, such as a specification is converted to an output such as a design. The activities here may be at a lower level than activities in a workflow model. They may perform transformations carried out by people or by computers.

- **3. A role/action model:** This means the roles of the people involved in the software process and the activities for which they are responsible.
- There are several various general models or paradigms of software development:
- **The waterfall approach:** This takes the above activities and produces them as separate process phases such as requirements specification, software design, implementation, testing, and so on. After each stage is defined, it is "signed off" and development goes onto the following stage.
- **Evolutionary development:** This method interleaves the activities of specification, development, and validation. An initial system is rapidly developed from a very abstract specification.
- **Formal transformation:** This method is based on producing a formal mathematical system specification and transforming this specification, using mathematical methods to a program. These transformations are 'correctness preserving.' This means that you can be sure that the developed programs meet its specification.
- **System assembly from reusable components:** This method assumes the parts of the system already exist.The system development process target on integrating these parts rather than developing them from scratch.

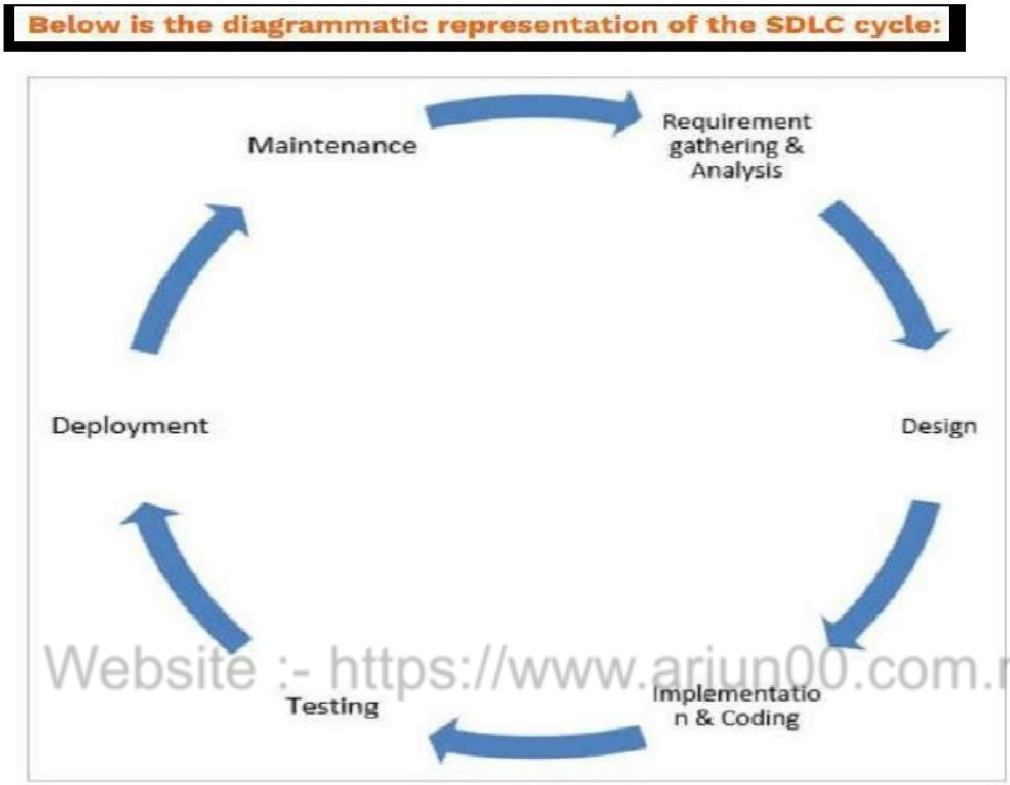
Software Development Life Cycles Models

- Software Development Life Cycles Models:
- 2.1 Build and fix model
- 2.2 The waterfall model
- 2.3 Prototyping model
- 2.4 Iterative enhancement model
- 2.5 Spiral model
- 2.6 Rapid application development model (RAD)
- 2.7 Selection criteria of a lifecycle model

- SDLC is a process that defines the various stages involved in the development of software for delivering a high-quality product. SDLC stages cover the complete life cycle of a software i.e. from inception to retirement of the product.

SDLC Cycle

- SDLC Cycle represents the process of developing software.



SDLC Phases

Given below are the various phases:

- Requirement gathering and analysis
- Design
- Implementation or coding
- Testing
- Deployment
- Maintenance

1) Requirement Gathering and Analysis

During this phase, all the relevant information is collected from the customer to develop a product as per their expectation. Any ambiguities must be resolved in this phase only.

2) Design

In this phase, the requirement gathered in the SRS document is used as an input and software architecture that is used for implementing system development is derived.

3) Implementation or Coding

Implementation/Coding starts once the developer gets the Design document. The Software design is translated into source code. All the components of the software are implemented in this phase.

4) Testing

Testing starts once the coding is complete and the modules are released for testing. In this phase, the developed software is tested thoroughly and any defects found are assigned to developers to get them fixed.

5) Deployment

Once the product is tested, it is deployed in the production environment or first UAT (User Acceptance testing) is done depending on the customer expectation.

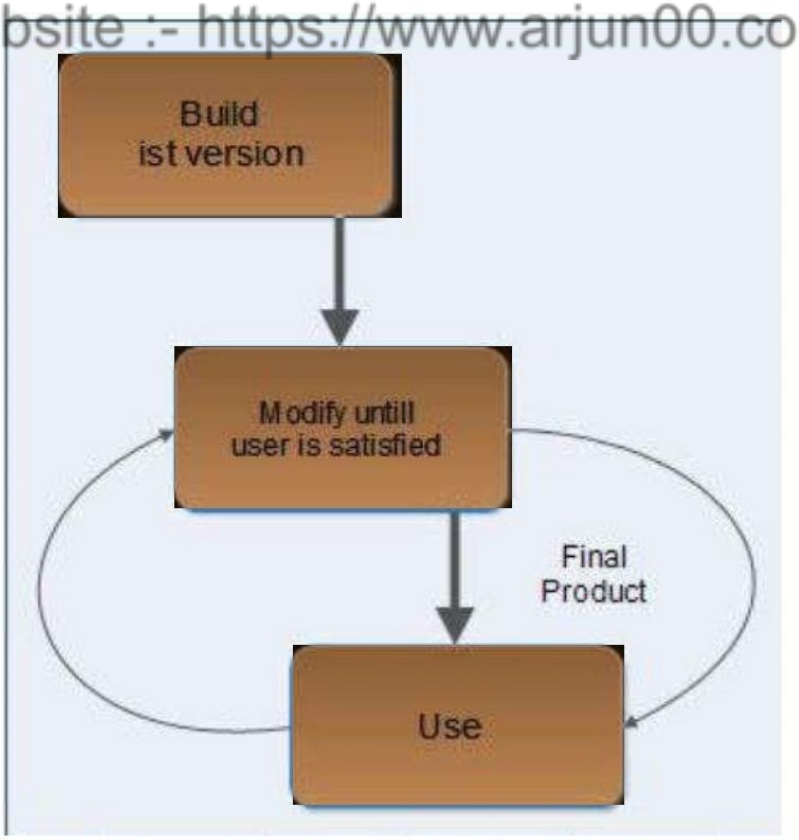
6) Maintenance

After the deployment of a product on the production environment, maintenance of the product i.e. if any issue comes up and needs to be fixed or any enhancement is to be done is taken care by the developers.

- **Software Development Life Cycle Models**
- A software life cycle model is a descriptive representation of the software development cycle. SDLC models might have a different approach but the basic phases and activity remain the same for all the models.

Build and fix model

- In the **build and fix model** (also referred to as an **ad hoc model**), the software is developed without any specification or design. An initial product is built, which is then repeatedly modified until it (software) satisfies the user. That is, the software is developed and delivered to the user. The user checks whether the desired functions are present. If not, then the software is changed according to the needs by adding, modifying or deleting functions. This process goes on until the user feels that the software can be used productively. However, the lack of design requirements and repeated modifications result in loss of acceptability of software. Thus, software engineers are strongly discouraged from using this development approach.



- This model includes the following two phases.
- **Build:** In this phase, the software code is developed and passed on to the next phase.
- **Fix:** In this phase, the code developed in the build phase is made error free. Also, in addition to the corrections to the code, the code is modified according to the user's requirements.

Table Advantages and Disadvantages of Build and Fix Model

Advantages	Disadvantages
▪ Requires less experience to execute or manage other than the ability to program. ▪ Suitable for smaller software. ▪ Requires less project planning.	▪ No real means is available of assessing the progress, quality, and risks. ▪ Cost of using this process model is high as it requires rework until user's requirements are accomplished. ▪ Informal design of the software as it involves unplanned procedure. ▪ Maintenance of these models is problematic.

Waterfall Model

- Waterfall model is the very first model that is used in SDLC. It is also known as the linear sequential model.
- In this model, the outcome of one phase is the input for the next phase. Development of the next phase starts only when the previous phase is complete.
- First, Requirement gathering and analysis is done. Once the requirement is freeze then only the System Design can start. Herein, the SRS document created is the output for the Requirement phase and it acts as an input for the System Design.
- In System Design Software architecture and Design, documents which act as an input for the next phase are created i.e. Implementation and coding.
- In the Implementation phase, coding is done and the software developed is the input for the next phase i.e. testing.
- In the testing phase, the developed code is tested thoroughly to detect the defects in the software. Defects are logged into the defect tracking tool and are retested once fixed. Bug logging, Retest, Regression testing goes on until the time the software is in go-live state.
- In the Deployment phase, the developed code is moved into production after the sign off is given by the customer.
- Any issues in the production environment are resolved by the developers which come under maintenance.

Advantages of the Waterfall Model:

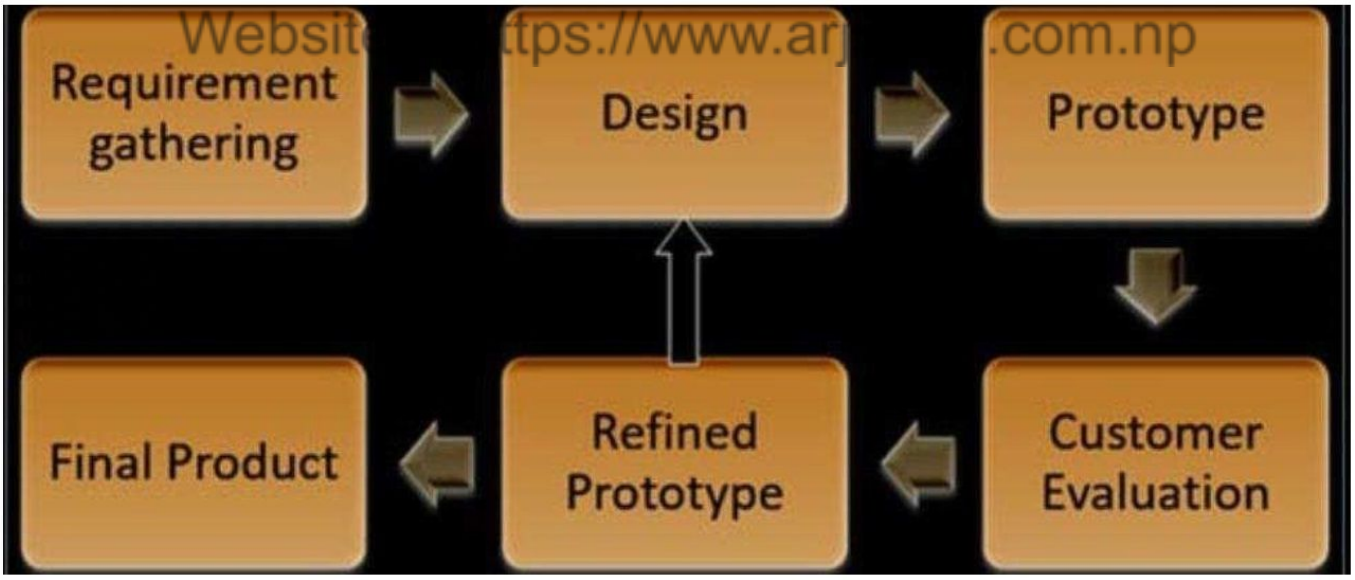
- Waterfall model is the simple model which can be easily understood and is the one in which all the phases are done step by step.
- Deliverables of each phase are well defined, and this leads to no complexity and makes the project easily manageable.

Disadvantages of Waterfall model:

- Waterfall model is time-consuming & cannot be used in the short duration projects as in this model a new phase cannot be started until the ongoing phase is completed.
- Waterfall model cannot be used for the projects which have uncertain requirement or wherein the requirement keeps on changing as this model expects the requirement to be clear in the requirement gathering and analysis phase itself and any change in the later stages would lead to cost higher as the changes would be required in all the phases.

Prototype Model

- The prototype model is a model in which the prototype is developed prior to the actual software.
- Prototype models have limited functional capabilities and inefficient performance when compared to the actual software. Dummy functions are used to create prototypes. This is a valuable mechanism for understanding the customers' needs.
- Software prototypes are built prior to the actual software to get valuable feedback from the customer. Feedbacks are implemented and the prototype is again reviewed by the customer for any change. This process goes on until the model is accepted by the customer.



- Once the requirement gathering is done, the quick design is created and the prototype which is presented to the customer for evaluation is built.
- Customer feedback and the refined requirement is used to modify the prototype and is again presented to the customer for evaluation. Once the customer approves the prototype, it is used as a requirement for building the actual software. The actual software is build using the Waterfall model approach.

Advantages of Prototype Model:

- Prototype model reduces the cost and time of development as the defects are found much earlier.
- Missing feature or functionality or a change in requirement can be identified in the evaluation phase and can be implemented in the refined prototype.
- Involvement of a customer from the initial stage reduces any confusion in the requirement or understanding of any functionality.

Disadvantages of Prototype Model:

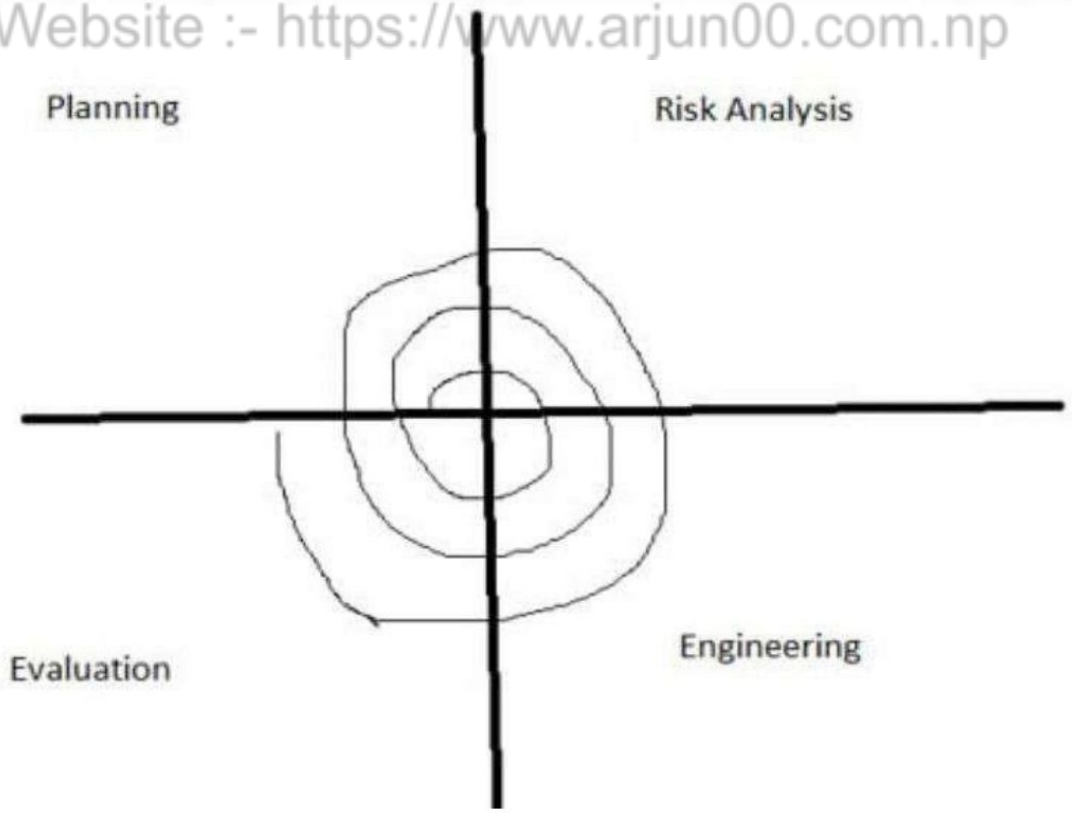
- Since the customer is involved in every phase, the customer can change the requirement of the end product which increases the complexity of the scope and may increase the delivery time of the product.

Spiral Model

- The Spiral Model includes iterative and prototype approach.
- Spiral model phases are followed in the iterations. The loops in the model represent the phase of the SDLC process i.e. the innermost loop is of requirement gathering & analysis which follows the Planning, Risk analysis, development, and evaluation. Next loop is Designing followed by Implementation & then testing.

Spiral Model has four phases:

- Planning
- Risk Analysis
- Engineering
- Evaluation



i) Planning:

- The planning phase includes requirement gathering wherein all the required information is gathered from the customer and is documented. Software requirement specification document is created for the next phase.

(ii) Risk Analysis:

- In this phase, the best solution is selected for the risks involved and analysis is done by building the prototype.
- **For Example**, the risk involved in accessing the data from a remote database can be that the data access rate might be too slow. The risk can be resolved by building a prototype of the data access subsystem.

(iii) Engineering:

- Once the risk analysis is done, coding and testing are done.

(iv) Evaluation:

- Customer evaluates the developed system and plans for the next iteration.

Advantages of Spiral Model:

- Risk Analysis is done extensively using the prototype models.
- Any enhancement or change in the functionality can be done in the next iteration.

Disadvantages of Spiral Model:

- The spiral model is best suited for large projects only.
- The cost can be high as it might take a large number of iterations which can lead to high time to reach the final product.

Iterative Incremental Model

- The iterative incremental model divides the product into small chunks.
- **For Example**, Feature to be developed in the iteration is decided and implemented. Each iteration goes through the phases namely Requirement Analysis, Designing, Coding, and Testing. Detailed planning is not required in iterations.
- Once the iteration is completed, a product is verified and is delivered to the customer for their evaluation and feedback. Customer's feedback is implemented in the next iteration along with the newly added feature.
- Hence, the product increments in terms of features and once the iterations are completed the final build holds all the features of the product.

Phases of Iterative & Incremental Development Model:

- Inception phase
- Elaboration Phase
- Construction Phase
- Transition Phase

(i) Inception Phase:

- Inception phase includes the requirement and scope of the Project.

(ii) Elaboration Phase:

- In the elaboration phase, the working architecture of a product is delivered which covers the risk identified in the inception phase and also fulfills the non-functional requirements.

(iii) Construction Phase:

- In the Construction phase, the architecture is filled in with the code which is ready to be deployed and is created through analysis, designing, implementation, and testing of the functional requirement.

(iv) Transition Phase:

- In the Transition Phase, the product is deployed in the Production environment.

Advantages of Iterative & Incremental Model:

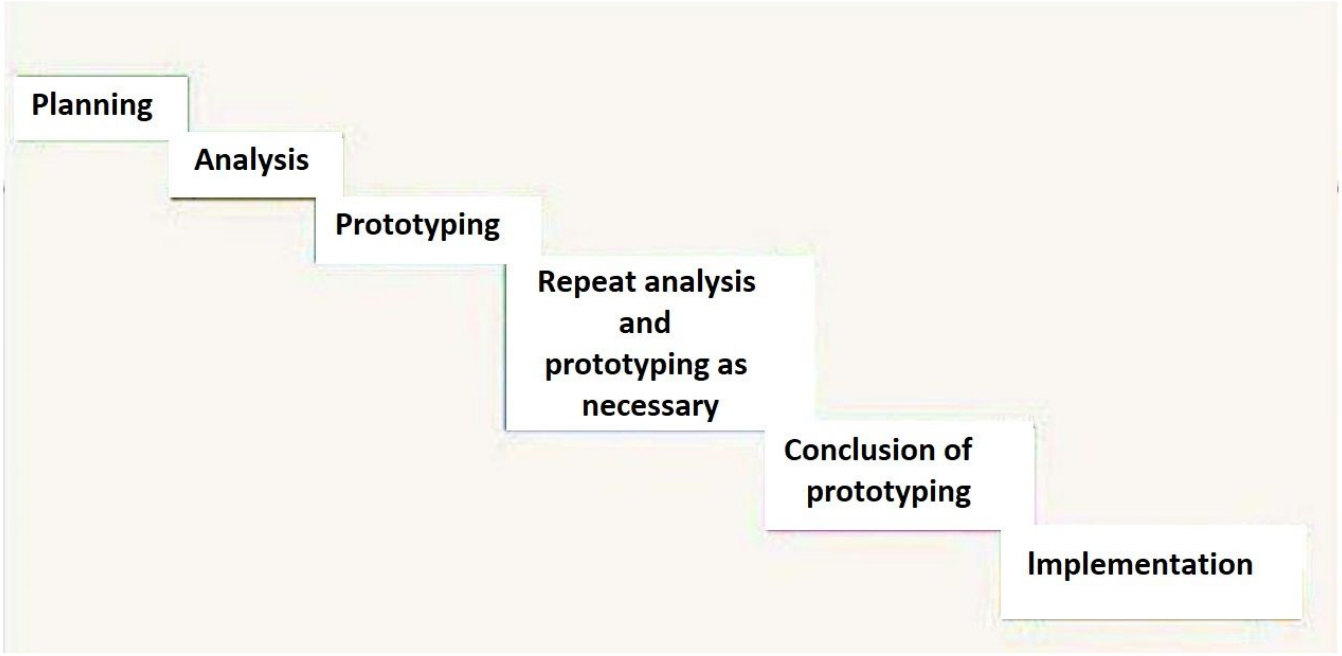
- Any change in the requirement can be easily done and would not cost as there is a scope of incorporating the new requirement in the next iteration.
- Risk is analyzed & identified in the iterations.
- Defects are detected at an early stage.
- As the product is divided into smaller chunks it is easy to manage the product.

Disadvantages of Iterative & Incremental Model:

- Complete requirement and understanding of a product are required to break down and build incrementally.

Rapid application development model (RAD)

- The rapid application development model emphasizes on delivering projects in small pieces. If the project is large, it is divided into a series of smaller projects. Each of these smaller projects is planned and delivered individually. Thus, with a series of smaller projects, the final project is delivered quickly and in a less structured manner. The major characteristic of the RAD model is that it focuses on the reuse of code, processes, templates, and tools.



The phases of RAD model are listed below.

- **Planning:** Inthis phase, the tasks and activities are planned. The derivables produced from this phase are project definition, project management procedures, and a work plan. **Project definition** determines and describes the project to be developed. **Project management procedure** describes processes for managing issues, scope, risk, communication, quality, and so on. **Work plan** describes the activities required for completing the project.

- **Analysis:** The requirements are gathered at a high level instead of at the precise set of detailed requirements level. Incase the user changes the requirements, RAD allows changing these requirements over a period of time. This phase determines plans for testing, training and implementation processes. Generally, the RAD projects are small in size, due to which high-level strategy documents are avoided.

- **Prototyping:** The requirements defined in the analysis phase are used to develop a prototype of the application. A final system is then developed with the help of the prototype. For this, it is essential to make decisions regarding technology and the tools required to develop the final system.
- **Repeat analysis and prototyping as necessary:** When the prototype is developed, it is sent to the user for evaluating its functioning. After the modified requirements are available, the prototype is updated according to the new set of requirements and is again sent to the user for analysis.

- **Conclusion of prototyping:** As a prototype is an iterative process, the project manager and user agree on a fixed number of processes. Ideally, three iterations are considered. After the third iteration, additional tasks for developing the software are performed and then tested. Last of all, the tested software is implemented.
- **Implementation:** The developed software, which is fully functioning, is deployed at the user's end.

Table Advantages and Disadvantages of RAD Model

Advantages	Disadvantages
▪ Deliverables are easier to transfer as high-level abstractions, scripts, and intermediate codes are used. ▪ Provides greater flexibility as redesign is done according to the developer. ▪ Results in reduction of manual coding due to code generators and code reuse. ▪ Encourages user involvement. ▪ Possibility of lesser defects due to prototyping in nature.	▪ Useful for only larger projects ▪ RAD projects fail if there is no commitment by the developers or the users to get software completed on time. ▪ Not appropriate when technical risks are high. This occurs when the new application utilizes new technology or when new software requires a high degree of interoperability with existing system. ▪ As the interests of users and developers can diverge from single iteration to next, requirements may not converge in RAD model.

Selection criteria of a lifecycle model

- Selecting the right SDLC is a process in itself that the organization can implement internally or consult for. There are some steps to get the right selection.
- STEP 1: Learn the about SDLC Models
- SDLCs are the same in their usage. In order to select the right SDLC, you should have enough experience and be familiar with the SDLCs that will be chosen and understand them correctly.

- STEP 2: Assess the needs of Stakeholders
- We must study the business domain, stakeholders concerns and requirements, business priorities, our technical capability and ability, and technology constraints to be able to choose the right SDLC against their selection criteria.
- STEP 3: Define the criteria
- Some of the selection criteria or arguments that you may use to select an SDLC are:
- Is the SDLC suitable for the size of our team and their skills?
- Is the SDLC suitable for the selected technology we use for implementing the solution?
- Is the SDLC suitable for client and stakeholders concerns and priorities?
- Is the SDLC suitable for the geographical situation (distributed team)?
- Is the SDLC suitable for the size and complexity of our software?
- Is the SDLC suitable for the type of projects we do?
- Is the SDLC suitable for our software engineering capability?
- Is the SDLC suitable for the project risk and quality insurance?
- STEP 4: Decide
- When you define the criteria and the arguments you need to discuss with the team, you will need to have a decision matrix and give each criterion a defined weight and score foreach option. After analyzing the results, you should documentthis decision in the project artifacts and share it with the related stakeholders.

- STEP 5: Optimize
- You can always optimize the SDLC during the project execution, you may notice upcoming changes do not fit with the selected SDLC, it is okay to align and cope with the changes. You can even make your own SDLC model which optimum for your organization or the type of projects you are involved in.

Software Project Management:

- 3.1 Activities in project management
- 3.2 Software project planning
- 3.3 Software project management plan
- 3.4 Software project scheduling and techniques
- 3.5 Software project team management and organization
- 3.6 Project estimation techniques
- 3.7 COCOMO model
- 3.8 Risk analysis and management
- 3.9 Risk management process
- 3.10 Software configuration management
- 3.11Software change management
- 3.12Version and release management

What is Project?

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

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.

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.

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:

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

Activities in project management

- Software Project Management consists of many activities, that includes planning of the project, deciding the scope of product, estimation of cost in different terms, scheduling of tasks, etc.

The list of activities are as follows:

- Project planning and Tracking
- Project Resource Management
- Scope Management
- Estimation Management
- Project Risk Management
- Scheduling Management
- Project Communication Management
- Configuration Management

- **1. Project Planning:** It is a set of multiple processes, or we can say that it a task that performed before the construction of the product starts.

- **2. Scope Management:** It describes the scope of the project. Scope management is important because it clearly defines what would do and what would not. Scope Management create the project to contain restricted and quantitative tasks, which may merely be documented and successively avoids price and time overrun.

3. Estimation management: This is not only about cost estimation because whenever we start to develop software, but we also figure out their size(line of code), efforts, time as well as cost.

- And if we talk about cost, it includes all the elements such as:
- Size of software
- Quality
- Hardware
- Communication
- Training
- Additional Software and tools
- Skilled manpower

4. Scheduling Management: Scheduling Management in software refers to all the activities to complete in the specified order and within time slotted to each activity. Project managers define multiple tasks and arrange them keeping various factors in mind.

5. Project Resource Management: In software Development, all the elements are referred to as resources for the project. It can be a human resource, productive tools, and libraries.

- Resource management includes:
- Create a project team and assign responsibilities to every team member
- Developing a resource plan is derived from the project plan.
- Adjustment of resources.

6. Project Risk Management: Risk management consists of all the activities like identification, analyzing and preparing the plan for predictable and unpredictable risk in the project.

- Several points show the risks in the project:
- The Experienced team leaves the project, and the new team joins it.
- Changes in requirement.
- Change in technologies and the environment.
- Market competition.

7. Project Communication Management: Communication is an essential factor in the success of the project. It is a bridge between client, organization, team members and as well as other stakeholders of the project such as hardware suppliers.

8. Project Configuration Management: Configuration management is about to control the changes in software like requirements, design, and development of the product.

Software Project Planning

- A Software Project is the complete methodology of programming advancement from requirement gathering to testing and support, completed by the execution procedures, in a specified period to achieve intended software product.
- Before starting a software project, it is essential to determine the tasks to be performed and properly manage allocation of tasks among individuals involved in the software development. Hence, planning is important as it results in effective software development.
- Project planning is an organized and integrated management process, which focuses on activities required for successful completion of the project. It prevents obstacles that arise in the project such as changes in projects or organization's objectives, non-availability of resources, and so on.
- Project planning also helps in better utilization of resources and optimal usage of the allotted time for a project. The other objectives of project planning are listed below.

1. It defines the roles and responsibilities of the project management team members.
2. It ensures that the project management team works according to the business objectives.
3. It checks feasibility of the schedule and user requirements.
4. It determines project constraints.

Tasks of Individuals Involved in Software Project

Senior Management	Project Management Team
• Approves the project, employ personnel, and provides resources required for the project. • Reviews project plan to ensure that it accomplishes the business objectives. • Resolves conflicts among the team members. • Considers risks that may affect the project so that appropriate measures can be taken to avoid them.	• Reviews the project plan and implements procedures for completing the project. • Manages all project activities. • Prepares budget and resource allocation plans. • Helps in resource distribution, project management, issue resolution, and so on. • Understands project objectives and finds ways to accomplish the objectives. • Devotes appropriate time and effort to achieve the expected results. • Selects methods and tools for the project.

Project Scheduling

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

1. Identify all the functions required to complete the project.
2. Break down large functions into small activities.
3. Determine the dependency among various activities.
4. Establish the most likely size for the time duration required to complete the activities.
5. Allocate resources to activities.
6. Plan the beginning and ending dates for different activities.
7. Determine the critical path. A critical way is the group of activities that decide the duration of the project.

Different Techniques of Project Scheduling

- Project Scheduling typically includes various techniques, an outline of each technique is provided below.

1. Mathematical Analysis

- Critical Path Method (CPM) and Program Evaluation and Review Technique (PERT) are the two most commonly used techniques by project managers. These methods are used to calculate the time span of the project through the scope of the project.

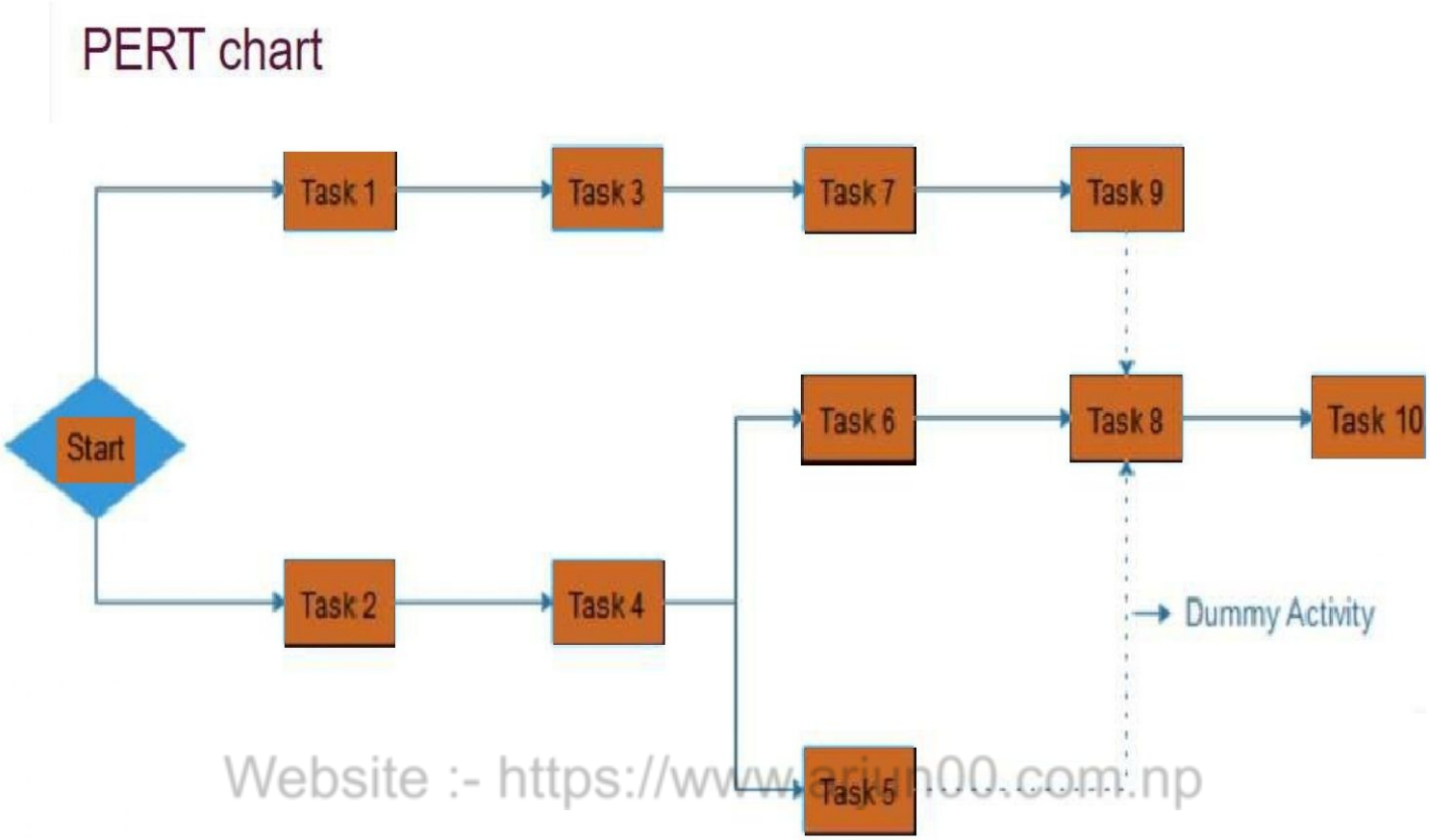
a. Critical Path Method

Every project's tree diagram has a critical path. The Critical Path Method estimates the maximum and minimum time required to complete a project. CPM also helps to identify critical tasks that should be incorporated into a project. Delivery time changes do not affect the schedule. The scope of the project and the list of activities necessary for the completion of the project are needed for using CPM. Next, the time taken by each activity is calculated. Then, all the dependent variables are identified. This helps in identifying and separating the independent variables. Finally, it adds milestones to the project.

b. Program Evaluation and Review Technique (PERT)

- PERT is a way to schedule the flow of tasks in a project and estimate the total time taken to complete it. This technique helps represent how each task is dependent on the other. To schedule a project using PERT, one has to define activities, arrange them in an orderly manner and define milestones. You can calculate timelines for a project on the basis of the level of confidence:

- Optimistic timing
- Most-likely timing
- Pessimistic timing



2. Duration Compression

- Duration compression helps to cut short a schedule if needed. It can adjust the set schedule by making changes without changing the scope in case, the project is running late. Two methodologies that can be applied: fast tracking and crashing.

a. Fast Tracking

- Fast-tracking is another way to use CPM. Fast-tracking finds ways to speed up the pace at which a project is being implemented by either simultaneously executing many tasks or by overlapping many tasks to each other.

b. Crashing

- Crashing deals with involving more resources to finish the project on time. For this to happen, you need spare resources to be available at your disposal. Moreover, all the tasks cannot be done by adding extra resources.

3. Simulation

- The expected duration of the project is calculated by using a different set of tasks in simulation. The schedule is created on the basis of assumptions, so it can be used even if the scope is changed or the tasks are not clear enough.

4. Resource-Leveling Heuristics

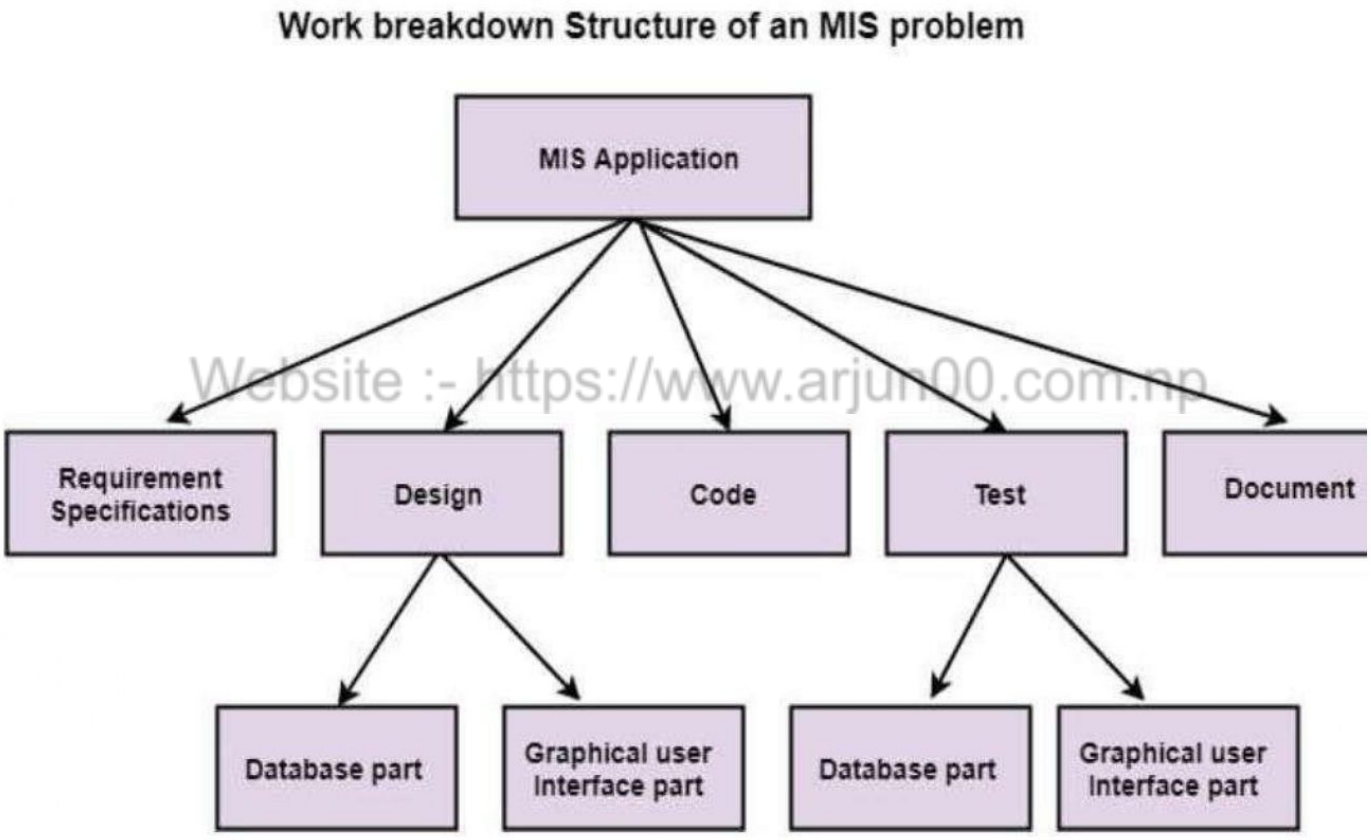
- Cutting the delivery time or avoiding under or overutilization of resources by making adjustments with the schedule or resources is called resource leveling heuristics. Dividing the tasks as per the available resources, so that no resource is under or over-utilized. The only demerit of this methodology is it may increase the project's cost and time.

5. Task List

- The task list is the simplest project scheduling technique of all the techniques available. Documented in a spreadsheet or word processor is the list of all possible tasks involved in a project. This method is simple and the most popular of all methods. It is very useful while implementing small projects. But for large projects with numerous aspects to consider task list is not a feasible method.

work breakdown structure

- The work breakdown structure formalism supports the manager to breakdown the function systematically after the project manager has broken down the purpose and constructs the work breakdown structure; he has to find the dependency among the activities. Dependency among the various activities determines the order in which the various events would be carried out.

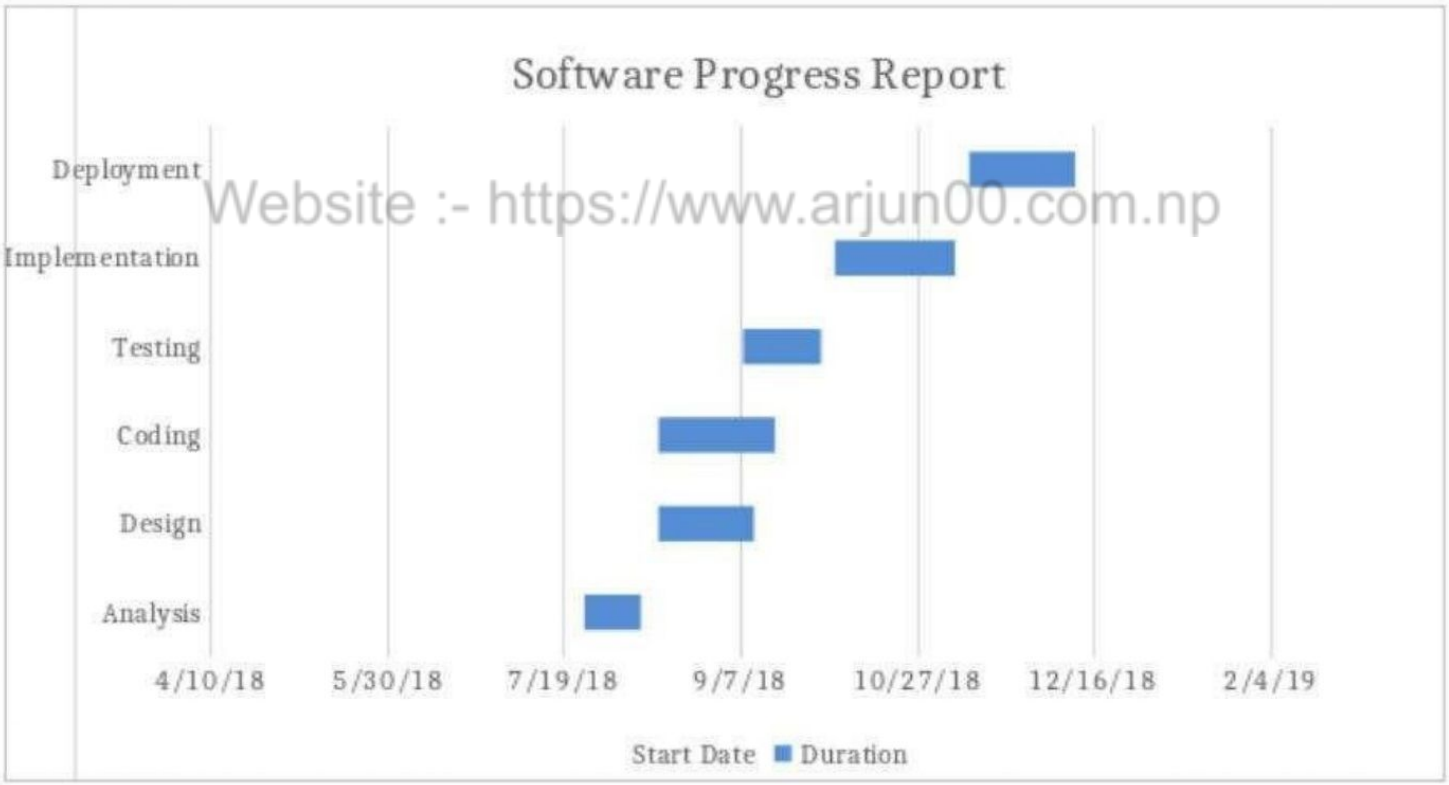


Team Management

- Team management includes the processes required to make the most effective use of the people involved with the project. The project team includes the project manager and the project staff who have been assigned with the responsibility to work on the project.

6. Gantt Chart

- For tracking progress and reporting purposes, the Gantt Chart is a visualization technique used in project management. It is used by project managers most of the time to get an idea about the average time needed to finish a project. A project schedule Gantt chart is a bar chart that represents key activities in sequence on the left vs time. Each task is represented by a bar that reflects the start and date of the activity, and therefore its duration.



- Team Management Process The major processes involved in team management are:
- ☐ Plan: Team identification, the process of identifying the skills and competencies required for carrying out the project activities and assign roles and responsibilities.
- ☐ Do: Team building, organizing the team and building their capacity to perform on the project, provide coaching and mentoring.
- ☐ Check: Evaluate team and individual performance, monitor skills, and motivation.
- ☐ Adapt: Improve team performance, build skills and set new targets

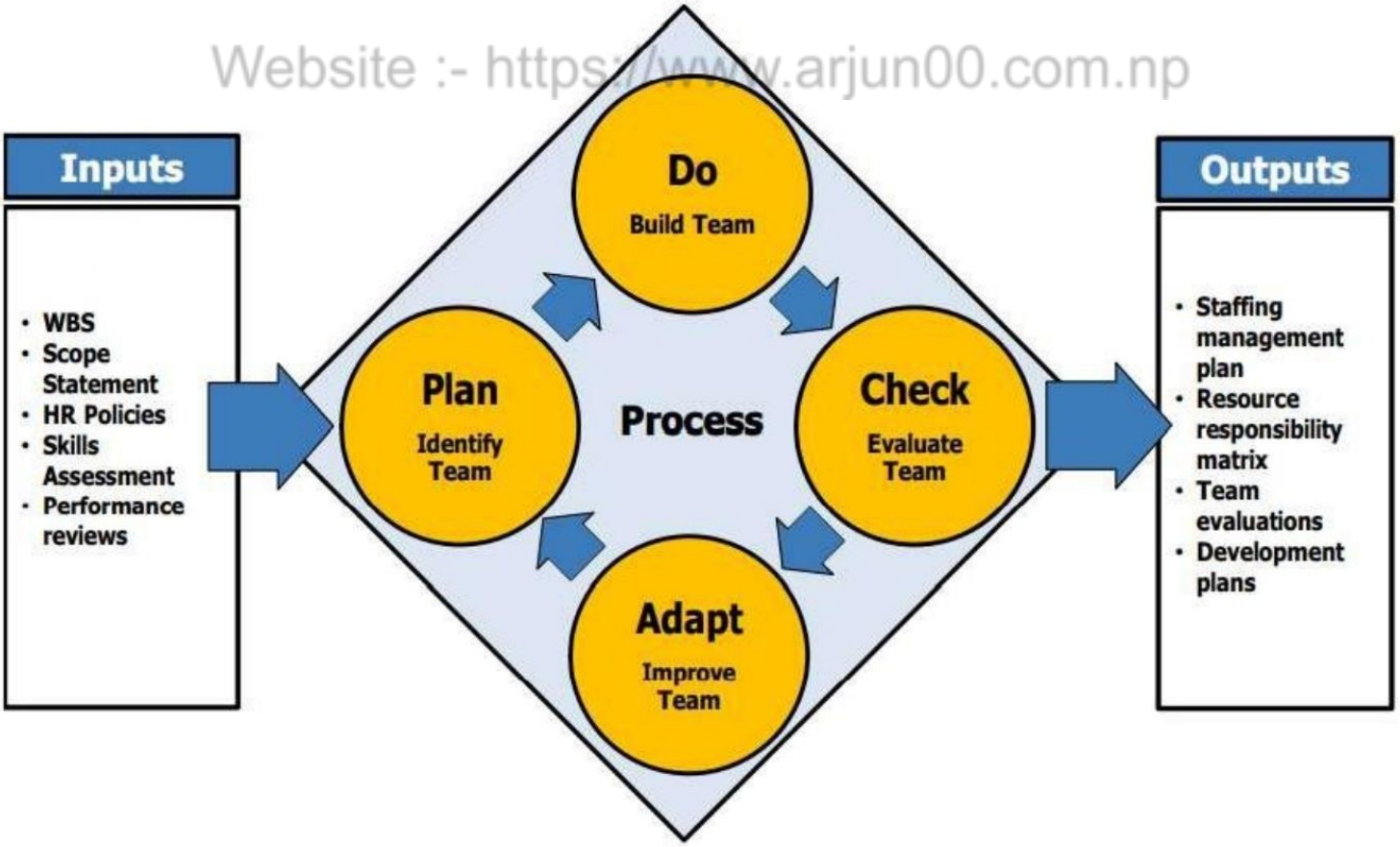


Figure 1 - Team IPO Chart

- Inputs: Inputs for the project team management include the following documents or sources of information:
- WBS
- Project Scope Statement
- HR organization policies
- Assessment of team skills
- Performance reviews

- Outputs:The project team will use the above information to develop four important documents for the project:
- Staffing management plan
- Resource responsibility matrix
- Team evaluations
- Development plans

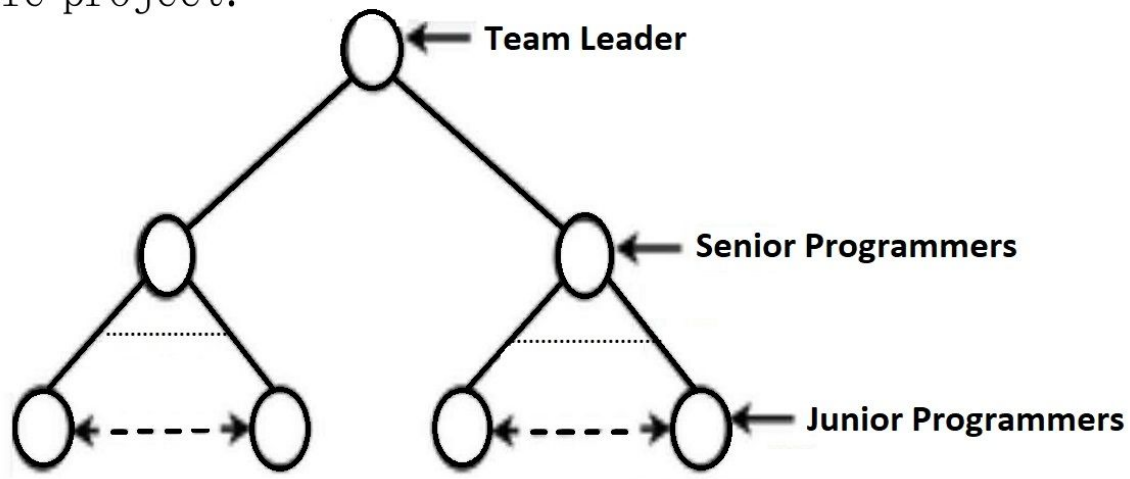
Software Project Team Organization

- There are many ways to organize the project team. Some important ways are as follows :

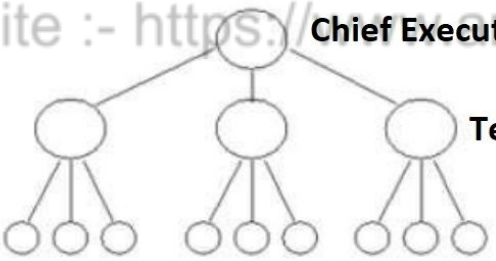
- Hierarchical team organization
- Chief-programmer team organization
- Matrix team, organization
- Egoless team organization
- Democratic team organization

- **Hierarchical team organization :**

In this, the people of organization at different levels following a tree structure. People at bottom level generally possess most detailed knowledge about the system. People at higher levels have broader appreciation of the whole project.



Hierarchical team organization



Large projects often distinguish levels of management:

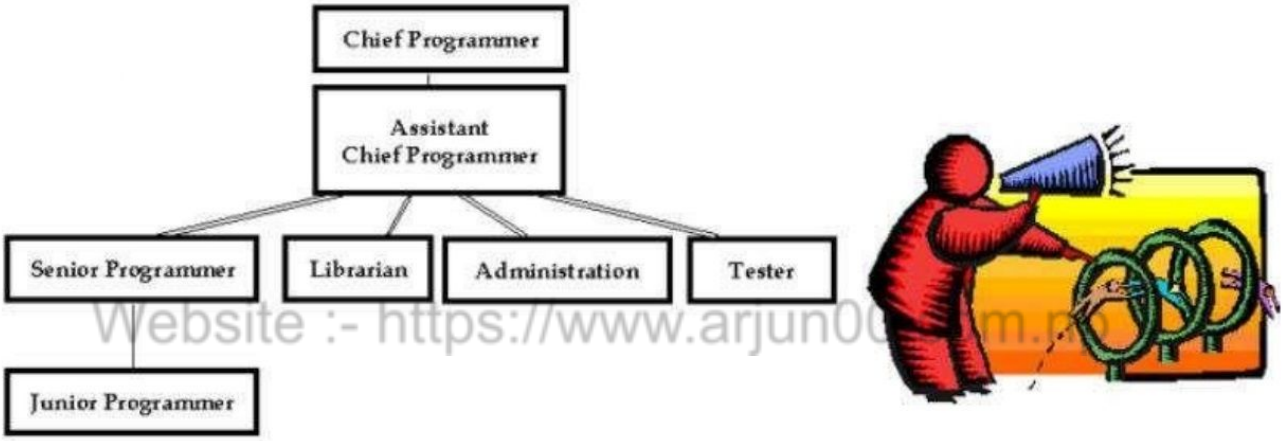
- Leaf nodes is where most development gets done; rest of tree is management
- Different levels do different kinds of work—a good programmer may not be a good manager
- Status and rewards depend on your level in the organization
- Works well when projects have high degree of certainty, stability and repetition
- But tends to produce overly positive reports on project progress, e.g.:
  - Bottom level: "We are having severe trouble implementing module X."
  - Level 1: "There are some problems with module X."
  - Level 2: "Progress is steady; I do not foresee any real problems."
  - Top: "Everything is proceeding according to our plan."

- **Chief-programmer team organization :**

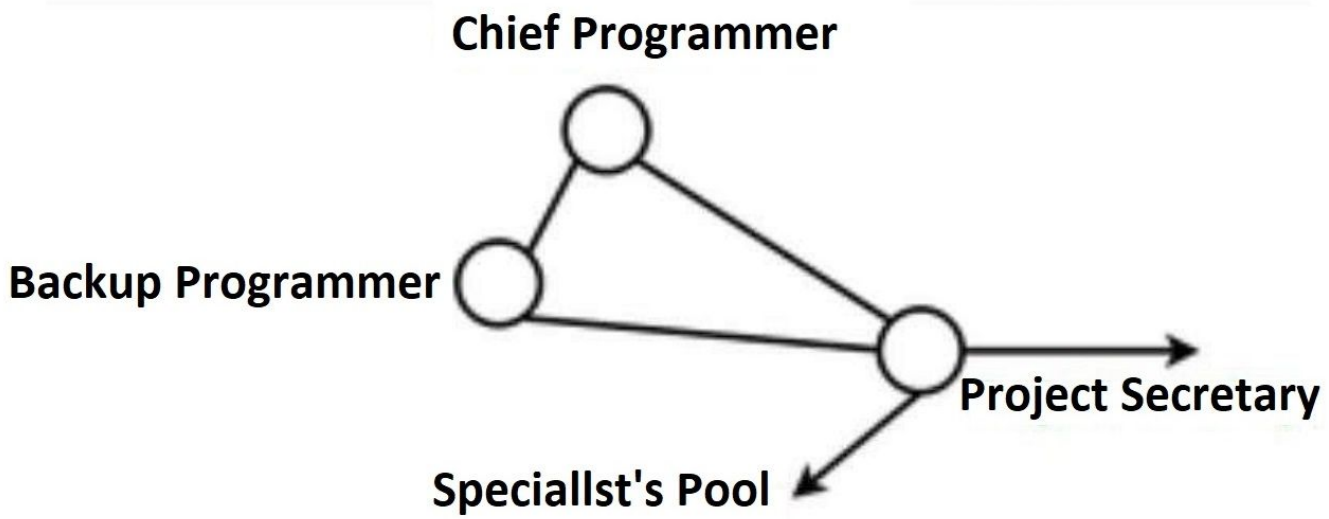
This team organization is composed of a small team consisting the following team members :

- **The Chief programmer :** It is the person who is actively involved in the planning, specification and design process and ideally in the implementation process as well.
- **The project assistant :** It is the closest technical co-worker of the chief programmer.
- **The project secretary :** It relieves the chief programmer and all other programmers of administration tools.
- **Specialists :** These people select the implementation language, implement individual system components and employ software tools and carry out tasks.

Chief Programmer Team



- *What do the graphics imply about this team structure?*
- **Chief programmer** makes all important decisions
  - Must be an expert analyst and architect, and a strong leader
- **Assistant Chief Programmer** can stand in for chief, if needed
- **Librarian** takes care of administration and documentation
- Additional developers have specialized roles



Matrix Team Organization :

- In matrix team organization, people are divided into specialist groups. Each group has a manager. Example of Metric team organization is as follows :

Egoless Team Organization :

- Egoless programming is a state of mind in which programmer are supposed to separate themselves from their product. In this team organization goals are set and decisions are made by group consensus. Here group, 'leadership' rotates based on tasks to be performed and differing abilities of members.

Matrix organization

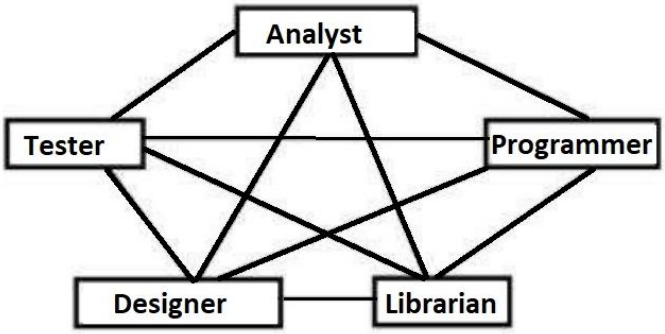
	real-time programming	graphics	databases	QA	testing
project C	X			X	X
project B	X		X	X	X
project A		X	X	X	X

- Organize people in terms of specialties
  - Matrix of projects and specialist groups
  - People from different departments allocated to software development, possibly part-time
- *Pros and cons?*
  - Project structure may not match organizational structure
  - Individuals have multiple bosses
  - Difficult to control a project's progress

Democratic Team Organization :

- It is quite similar to the egoless team organization, but one member is the team leader with some responsibilities :
  - Coordination
  - Final decisions, when consensus cannot be reached.

Democratic or Open structured teams



- A "grass roots" anti-elitist style of team organization
  - Egoless: group owns the documents & code (not individuals)
  - All decisions are based on team consensus
  - Depends on total cooperation of its members
  - Requires clear structure for the way the team interacts
  - Functional roles (e.g. moderator, recorder) rotate among team members
  - A technical leader has external responsibility and resolves issues when team doesn't reach consensus

Software Engineering | Project size estimation techniques

- Estimation of the size of the software is an essential part of Software Project Management. It helps the project manager to further predict the effort and time which will be needed to build the project. Various measures are used in project size estimation. Some of these are:
  - Lines of Code
  - Number of entities in ER diagram
  - Total number of processes in detailed data flow diagram
  - Function points

1. Lines of Code (LOC): As the name suggests, LOC count the total number of lines of source code in a project. The units of LOC are:

- KLOC- Thousand lines of code
- NLOC- Non-comment lines of code
- KDSI- Thousands of delivered source instruction
- The size is estimated by comparing it with the existing systems of the same kind. The experts use it to predict the required size of various components of software and then add them to get the total size.

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

- **Advantage**
  - Size estimation can be done during the initial stages of planning.
  - The number of entities is independent of the programming technologies used.
- **Disadvantages:**
  - No fixed standards exist. Some entities contribute more project size than others.
  - Just like FPA, it is less used in the cost estimation model. Hence, it must be converted to LOC.

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

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

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

- **4. Function Point Analysis:** In this method, the number and type of functions supported by the software are utilized to find FPC(function point count).The steps in function point analysis are:
  - Count the number of functions of each proposed type.
  - Compute the Unadjusted Function Points(UFP).
  - FindTotal Degree of Influence(TDI).
  - Compute Value Adjustment Factor(VAF).
  - Find the Function Point Count(FPC).

Software Engineering | 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:

  - **Effort:** Amount of labor that will be required to complete a task. It is measured in person-months units.
  - **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 tobe 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:
  - Basic COCOMO Model
  - Intermediate COCOMO Model
  - 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 and **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.

- Risk analysis and management 3.9 Risk management process 3.10 Software configuration management 3.11Software change management 3.12Version and release management

---

What is Risk?

- "Tomorrow problems are today's risk." Hence, a clear definition of a "risk" is a problem that could cause some loss or threaten the progress of the project, but which has not happened yet.

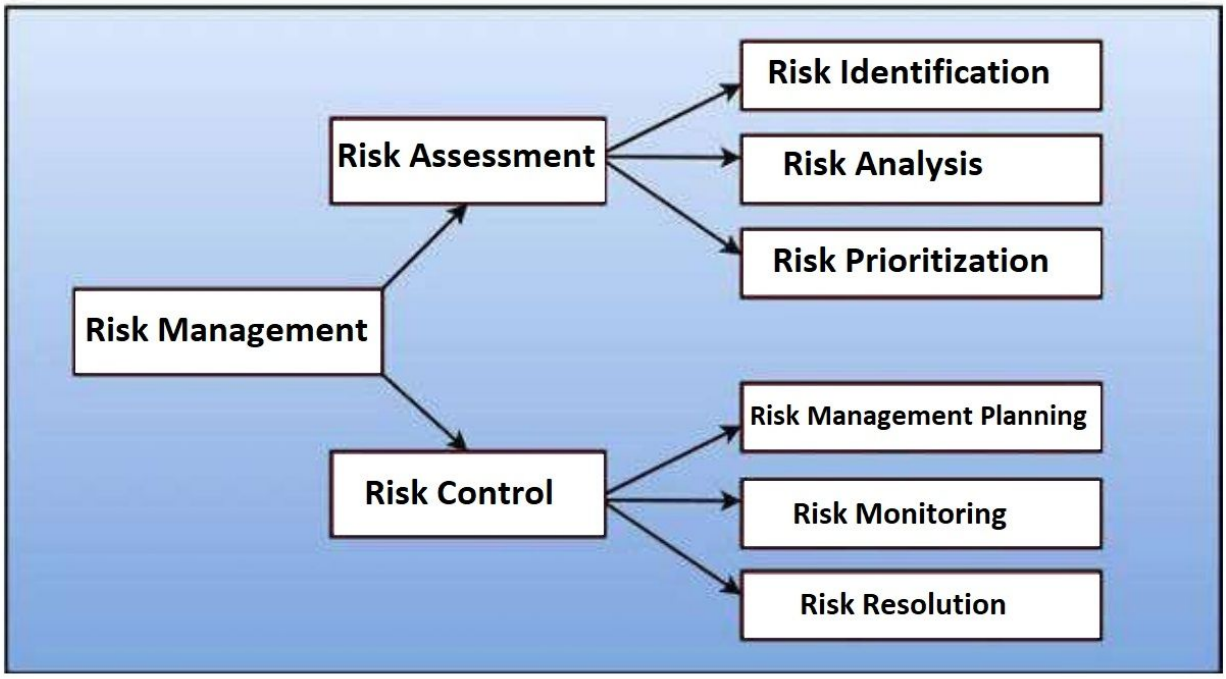
Risk Management

- A software project can be concerned with a large variety of risks. In order to be adept to systematically identify the significant risks which might affect a software project, it is essential to classify risks into different classes. The project manager can then check which risks from each class are relevant to the project.
- There are three main classifications of risks which can affect a software project:
  - Project risks
  - Technical risks
  - Business risks

- **1. Project risks:** Project risks concern differ forms of budgetary, schedule, personnel, resource, and customer-related problems. A vital project risk is schedule slippage
- **2. Technical risks:** Technical risks concern potential method, implementation, interfacing, testing, and maintenance issue.
- **3. Business risks:** This type of risks contain risks of building an excellent product that no one need, losing budgetary or personnel commitments, etc.

- Risk Management Activities

- **Risk management consists of three main activities, as shown in fig:**



Risk Management Activities

- Risk Assessment
- The objective of risk assessment is to division the risks in the condition of their loss, causing potential. For risk assessment, first, every risk should be rated in two methods:
  - The possibility of a risk coming true (denoted as r).
  - The consequence of the issues relates to that risk (denoted as s).
  - Based on these two methods, the priority of each risk can be estimated:
  - $p = r * s$
- Where p is the priority with which the risk must be controlled, r is the probability of the risk becoming true, and s is the severity of loss caused due to the risk becoming true. If all identified risks are set up, then the most likely and damaging risks can be controlled first, and more comprehensive risk abatement methods can be designed for these risks.

- **Perform qualitative risk analysis**
- It is the process of prioritizing risks for further analysis of project risk or action by combining and assessing their probability of occurrence and impact. It helps managers to lessen the uncertainty level and concentrate on high priority risks.

- Risk Control
- It is the process of managing risks to achieve desired outcomes. After all, the identified risks of a plan are determined; the project must be made to include the most harmful and the most likely risks. Different risks need different containment methods.

- **Risk Leverage:** To choose between the various methods of handling risk, the project plan must consider the amount of controlling the risk and the corresponding reduction of risk. For this, the risk leverage of the various risks can be estimated.
- Risk leverage is the variation in risk exposure divided by the amount of reducing the risk.
- **Risk leverage = (risk exposure before reduction - risk exposure after reduction) / (cost of reduction)**

1. Risk planning: The risk planning method considers each of the key risks that have been identified and develop ways to maintain these risks.

2. Risk Monitoring: Risk monitoring is the method king that your assumption about the product, process, and business risks has not changed.

- **What is Risk Analysis?**
- **Risk Analysis** in project management is a sequence of processes to identify the factors that may affect a project's success. These processes include risk identification, analysis of risks, risk management and control, etc. Proper risk analysis helps to control possible future events that may harm the overall project. It is more of a pro-active than a reactive process.

How to Manage Risk?

- Risk Management in Software Engineering primarily involves following activities:

Plan risk management

- It is the procedure of defining how to perform risk management activities for a project.

Risk Identification

- It is the procedure of determining which risk may affect the project most. This process involves documentation of existing risks.

Quantitative risk analysis

- It is the procedure of numerically analyzing the effect of identified risks on overall project objectives. In order to minimize the project uncertainty, this kind of analysis are quite helpful for decision making.

- **Plan risk responses**
- To enhance opportunities and to minimize threats to project objectives plan risk response is helpful. It addresses the risks by their priority, activities into the budget, schedule, and project management plan.
- **Control Risks**
- Control risk is the procedure of tracking identified risks, identifying new risks, monitoring residual risks and evaluating risk.

Software Configuration Management

- When we develop software, the product (software) undergoes many changes in their maintenance phase; we need to handle these changes effectively.
- Several individuals (programs) works together to achieve these common goals. This individual produces several work product (SC Items) e.g., Intermediate version of modules or test data used during debugging, parts of the final product.
- The elements that comprise all information produced as a part of the software process are collectively called a software configuration.
- As software development progresses, the number of Software Configuration elements (SCI's) grow rapidly.

- **These are handled and controlled by SCM. This is where we require software configuration management.**
- A configuration of the product refers not only to the product's constituent but also to a particular version of the component.
- Therefore, SCM is the discipline which
- Identify change
- Monitor and control change
- Ensure the proper implementation of change made to the item.
- Auditing and reporting on the change made.
- Configuration Management (CM) is a technic of identifying, organizing, and controlling modification to software being built by a programming team.

3.11 Software change management

- Change Management in software development refers to the transition from an existing state of the software product to another improved state of the product. It controls, supports, and manages changes to artifacts, such as code changes, process changes, or documentation changes. Where CCP (Change Control Process) mainly identifies, documents, and authorizes changes to a software application.

- **Process of Change Management :**

- When any software application/product goes for any changes in an IT environment, it undergoes a series of sequential processes as follows:

- Creating a request for change
- Reviewing and assessing a request for change
- Planning the change
- Testing the change
- Creating a change proposal
- Implementing changes
- Reviewing change performance
- Closing the process

- **Importance of Change Management :**
  - For improving performance
  - For increasing engagement
  - For enhancing innovation
  - For including new technologies
  - For implementing new requirements
  - For reducing cost

- **Key points to be considered during Change Management :**
- Reason of change
- Result of change
- The portion to be changed
- Person will change
- Risks involved in change
- Alternative to change
- Resources required for change
- Relationship between changes

3.12 Version and release management

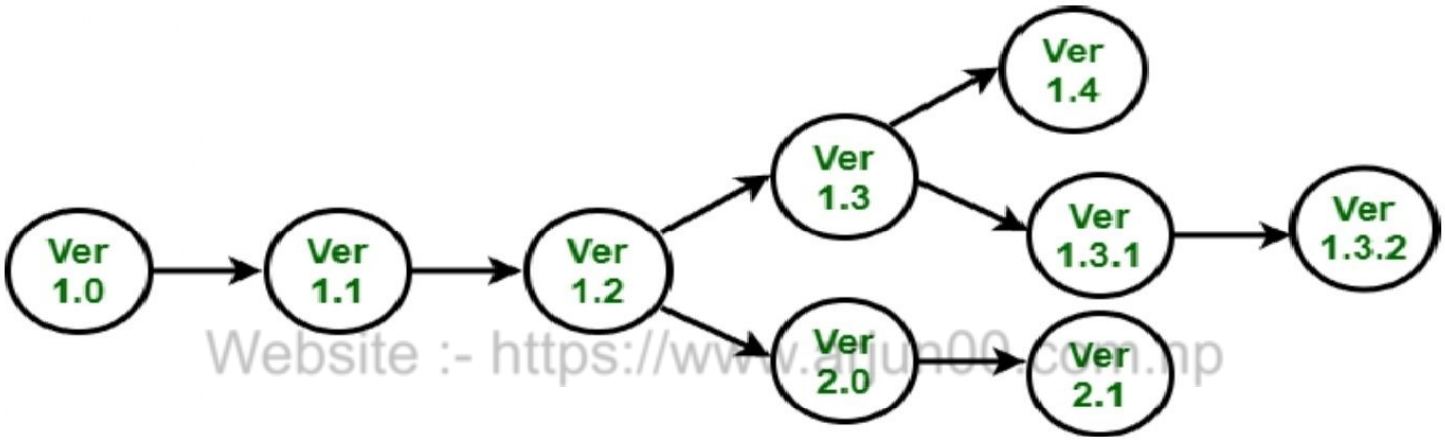
- The process involved in version and release management are concerned with identifying and keeping track of the versions of a system. Versions managers devise procedures to ensure that versions of a system may be retrieved when required and are not accidentally changedby the development team. For products, version managers work with marketing staff and for custom systems with customers, to plan when new releases of a system should be created not distributed for deployment.
- Some versions may be functionally equivalent but designed for different hardware or software configuration. Versions with only small differences are sometimes called **variants.**
- A system release may be a version that's distributed to customers. Each system release should either include new functionality or should be intended for a special hardware platform. There are normally many more versions of a system than release. Versions are created with an organization for internal development or testing and are not intended for release to customers.

- **Version Identification :**

  To create a specific version of a system, you've got to specify the versions of the system components that ought to be included in it. In a large software system, there are hundreds to software components, each of which may exist in several different versions.
- There must therefore be an unambiguous way to identify each component version to ensure that the right components are included in the system. Three basic techniques are used for components version identification

- **Version Numbering :**

In version numbering scheme, a version number is added to the components or system name. If the first version is called 1.0, subsequent versions are 1.1, 1.2 and so on. At some stage, a new release is created (release 2.0) and process start again at version 2.1. The scheme is linear, based on the assumption that system versions are created in sequence. Most version management tools such as RCS and CVS support this approach to version identification.



- **Attribute Based Identification :**

If each version is identified by a unique set of attributes, it is easy to add new versions, that are derived from any of existing versions. These are identified using unique set of attribute values.

- **Change Oriented Identification :**

Each component is known as as in attribute-based identification but is additionally related to one or more change requests. That is, it is assumed that each version of component has been created in response to one or more change requests. Component version is identified by set of change requests that apply to components.

Software Requirement Analysis & Specification:

Requirement Engineering

- Requirement Engineering is the process of defining, documenting and maintaining the requirements. It is a process of gathering and defining service provided by the system. Requirements Engineering Process consists of the following main activities:
  - Requirements elicitation
  - Requirements specification
  - Requirements verification and validation
  - Requirements management

- **Requirements elicitation** is perhaps the most difficult, most error-prone and most communication intensive software development. It can be successful only through an effective customer-developer partnership. It is needed to know what the users really need.
- There are a number of requirements elicitation methods. Few of them are listed below –
  - Requirement Elicitation Techniques
  - Requirements Elicitation is the process to find out the requirements for an intended software system by communicating with client, end users, system users and others who have a stake in the software system development.
  - There are various ways to discover requirements

  - Interviews
  - Interviews are strong medium to collect requirements. Organization may conduct several types of interviews such as:
  - Structured (closed) interviews, where every single information to gather is decided in advance, they follow pattern and matter of discussion firmly.
  - Non-structured (open) interviews, where information to gather is not decided in advance, more flexible and less biased.
  - Oral interviews
  - Written interviews
  - One-to-one interviews which are held between two persons across the table.
  - Group interviews which are held between groups of participants. They help to uncover any missing requirement as numerous people are involved.

- Brainstorming

- An informal debate is held among various stakeholders and all their inputs are recorded for further requirements analysis.
- It is a group technique
- It is intended to generate lots of new ideas hence providing a platform to share views
- A highly trained facilitator is required to handle group bias and group conflicts.
- Every idea is documented so that everyone can see it.
- Finally, a document is prepared which consists of the list of requirements and their priority if possible.

Use Case Approach:

- This technique combines text and pictures to provide a better understanding of the requirements.

The use cases describe the 'what', of a system and not 'how'. Hence, they only give a functional view of the system.
The components of the use case design includes three major things - Actor, Use cases, use case diagram.

- **Actor -**

It is the external agent that lies outside the system but interacts with it in some way. An actor maybe a person, machine etc. It is represented as a stick figure. Actors can be primary actors or secondary actors.

- Primary actors - It requires assistance from the system to achieve a goal.
- Secondary actor - It is an actor from which the system needs assistance.

- **Use cases -**
  They describe the sequence of interactions between actors and the system. They capture who (actors) do what (interaction) with the system. A complete set of use cases specifies all possible ways to use the system.
- **Use case diagram -**
  A use case diagram graphically represents what happens when an actor interacts with a system. It captures the functional aspect of the system.
  - A stick figure is used to represent an actor.
  - An oval is used to represent a use case.
  - A line is used to represent a relationship between an actor and a use case.
- Requirement analysis 4.3.1. Data flow diagram 4.3.2 Data dictionary 4.3.3 Entity-Relationship diagram 4.3.4 Software prototyping

- Requirement Analysis, also known as Requirement Engineering, is the process of defining user expectations for a new software being built or modified.
- Some of the technique for requirement analysis are:

- **Data Flow Diagrams:** Data Flow Diagrams (DFDs) are used widely for modeling the requirements. DFD shows the flow of data through a system. The system may be a company, an organization, a set of procedures, a computer hardware system, a software system, or any combination of the preceding. The DFD is also known as a data flow graph or bubble chart.

- **Data Dictionaries:** Data Dictionaries are simply repositories to store information about all data items defined in DFDs. At the requirements stage, the data dictionary should at least define customer data items, to ensure that the customer and developers use the same definition and terminologies.
- **Entity-Relationship Diagrams:** Another tool for requirement specification is the entity-relationship diagram, often called an "***E-R diagram.***" It is a detailed logical representation of the data for the organization and uses three main constructs i.e. data entities, relationships, and their associated attributes.
- Requirement documentation 4.4.1 Nature of SRS 4.4.2 Characteristics of a good SRS 4.4.3 Organization of SRS

- A Software Requirements Specification (SRS) is a document that describes the nature of a project, software or application. In simple words, SRS document is a manual of a project provided it is prepared before you kick-start a project/application.
- **Nature of Software Requirement Specification (SRS):**
- **The basic issues that SRS writer shall address are the following:**

1. Functionality: What the software is supposed to do?

2. External Interfaces: How does the software interact with people, system's hardware, other hardware and other software?

- **3. Performance:** What is the speed, availability, response time, recovery time etc.

4. Attributes: What are the considerations for portability, correctness, maintainability, security, reliability etc.

5. Design Constraints Imposed on an Implementation: Are there any required standards in effect, implementation language, policies for database integrity, resource limits, operating environment etc.

Characteristics of good SRS

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

- **2. Completeness:**
- Completeness of SRS indicates every sense of completion including the numbering of all the pages, resolving the to be determined parts to as much extent as possible as well as covering all the functional and non-functional requirements properly.
- **3. Consistency:**

Requirements in SRS are said to be consistent if there are no conflicts between any set of requirements. Examples of conflict include differences in terminologies used at separate places, logical conflicts like time period of report generation, etc.

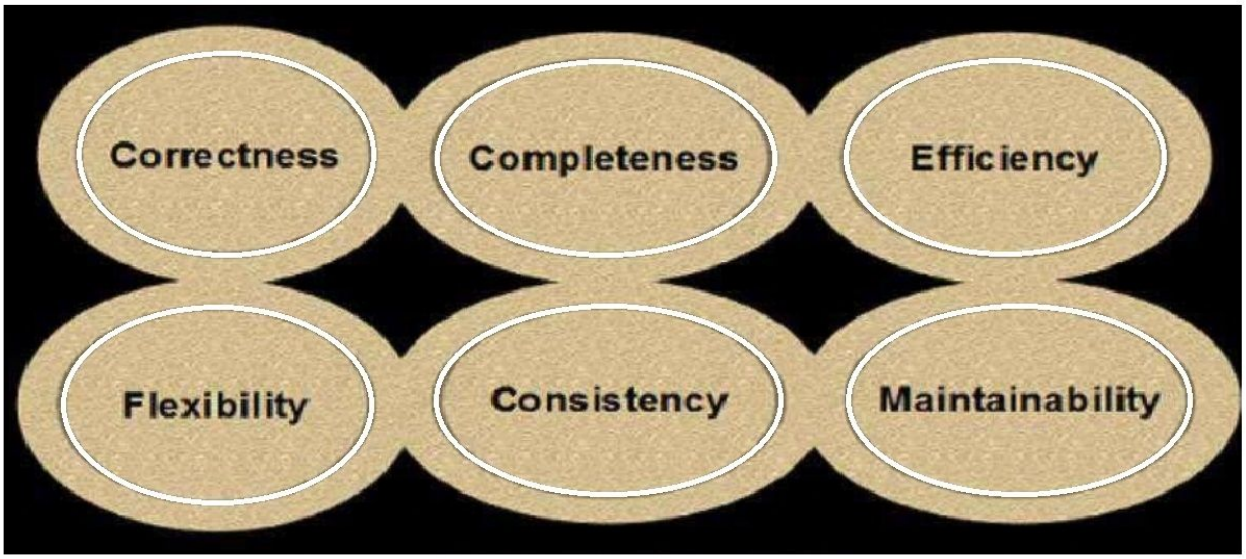
- **4.Unambiguousness:** SRS is unambiguous when every fixed requirement has only one interpretation. This suggests that each element is uniquely interpreted. In case there is a method used with multiple definitions, the requirements report should determine the implications in the SRS so that it is clear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

- **5. Ranking for importance and stability:** The SRS is ranked for importance and stability if each requirement in it has an identifier to indicate either the significance or stability of that particular requirement.
- **6. Modifiability:** SRS should be made as modifiable as likely and should be capable of quickly obtain changes to the system to some extent. Modifications should be perfectly indexed and cross-referenced.
- **7.Verifiability:** SRS is correct when the specified requirements can be verified with a cost-effective system to check whether the final software meets those requirements.The requirements are verified with the help of reviews.
- **8.Traceability:** The SRS is traceable if the origin of each of the requirements is clear and if it facilitates the referencing of each condition in future development or enhancment documentation.
- **9. Design Independence:** There should be an option to select from multiple design alternatives for the final system. More specifically, the SRS should not contain any implementation details.
- **10.Testability:** An SRS should be written in such a method that it is simple to generate test cases and test plans from the report.
- **11. Understandable by the customer:** An end user may be an expert in his/her explicit domain but might not be trained in computer science. Hence, the purpose of formal notations and symbols should be avoided too as much extent as possible.The language should be kept simple and clear.

Software Design:

- 5.1 Objectives of design
- 5.2 Design framework
- 5.3 Software design models
- 5.4 Design process
- 5.5 Architecture design
- 5.6 Low level design
- 5.7 Coupling and cohesion
- 5.8 Software design strategies
- 5.9 Function oriented design
- 5.10 Object oriented design
- 5.11 Function oriented design Vs Object oriented design

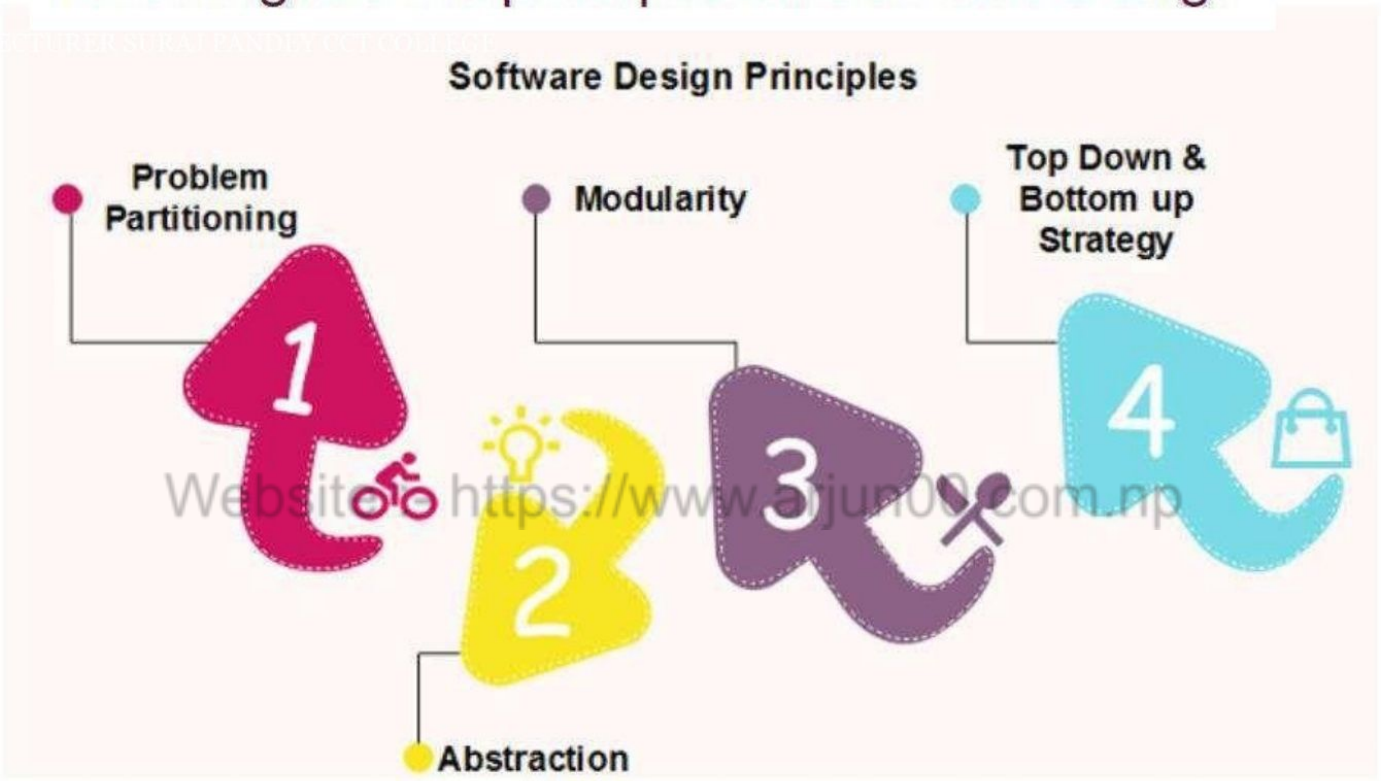
- Software design is a process to transform user requirements into some suitable form, which helps the programmer in software coding and implementation.
- Objectives of Software Design
- Following are the purposes of Software design:



Objectives of Software Design

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

Following are the principles of Software Design



- Problem Partitioning
- For small problem, we can handle the entire problem at once but for the significant problem, divide the problems and conquer the problem it means to divide the problem into smaller pieces so that each piece can be captured separately.
- For software design, the goal is to divide the problem into manageable pieces.
- Benefits of Problem Partitioning
- Software is easy to understand
- Software becomes simple
- Software is easy to test
- Software is easy to modify
- Software is easy to maintain
- Software is easy to expand

- Abstraction
- An abstraction is a tool that enables a designer to consider a component at an abstract level without bothering about the internal details of the implementation. Abstraction can be used for existing element as well as the component being designed.
- Here, there are two common abstraction mechanisms
- Functional Abstraction
- Data Abstraction
- Functional Abstraction
- A module is specified by the method it performs.
- The details of the algorithm to accomplish the functions are not visible to the user of the function.
- Functional abstraction forms the basis for **Function oriented design approaches**.
- Data Abstraction
- Details of the data elements are not visible to the users of data. Data Abstraction forms the basis for **Object Oriented design approaches**.
- Modularity
- Modularity specifies to the division of software into separate modules which are differently named and addressed and are integrated later on in to obtain the completely functional software. It is the only property that allows a program to be intellectually manageable. Single large programs are difficult to understand and read due to a large number of reference variables, control paths, global variables, etc.

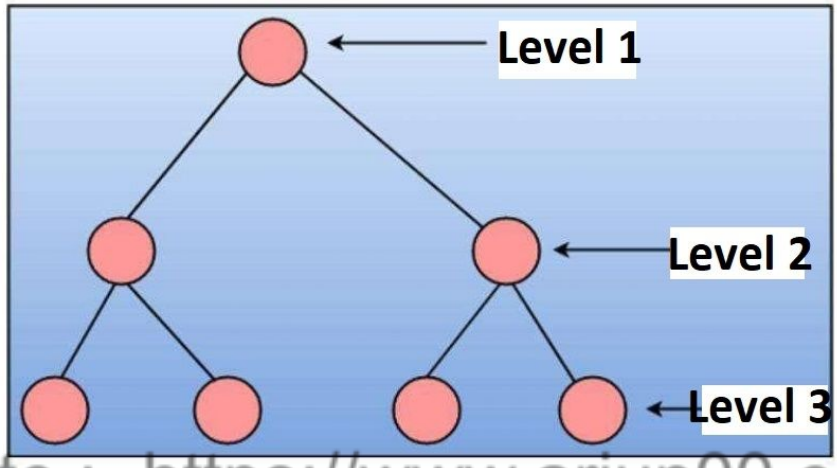
- There are several advantages of Modularity
- It allows large programs to be written by several or different people
- It encourages the creation of commonly used routines to be placed in the library and used by other programs.
- It simplifies the overlay procedure of loading a large program into main storage.
- It provides more checkpoints to measure progress.
- It provides a framework for complete testing, more accessible to test
- It produced the well designed and more readable program.
- **Disadvantages of Modularity**
- There are several disadvantages of Modularity
- Execution time maybe, but not certainly, longer
- Storage size perhaps, but is not certainly, increased
- Compilation and loading time may be longer
- Inter-module communication problems may be increased
- More linkage required, run-time may be longer, more source lines must be written, and more documentation has to be done

- Modular Design
- Modular design reduces the design complexity and results in easier and faster implementation by allowing parallel development of various parts of a system. We discuss a different section of modular design in detail in this section:
- **1. Functional Independence:** Functional independence is achieved by developing functions that perform only one kind of task and do not excessively interact with other modules. Independence is important because it makes implementation more accessible and faster. The independent modules are easier to maintain, test, and reduce error propagation and can be reused in other programs as well. Thus, functional independence is a good design feature which ensures software quality.
- **It is measured using two criteria:**
- **Cohesion:** It measures the relative function strength of a module.
- **Coupling:** It measures the relative interdependence among modules.

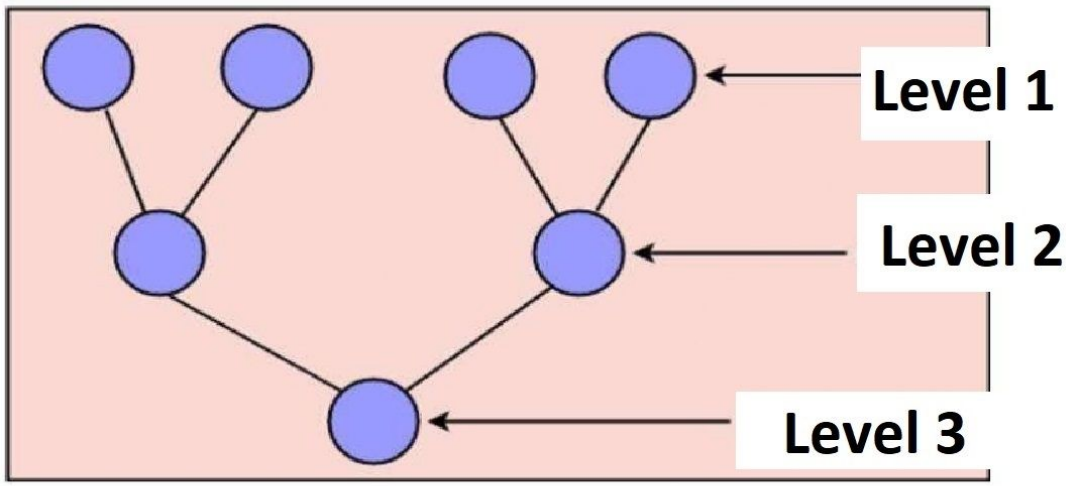
- **2. Information hiding:** The fundamental of Information hiding suggests that modules can be characterized by the design decisions that protect from the others, i.e., In other words, modules should be specified that data include within a module is inaccessible to other modules that do not need for such information.
- Strategy of Design
- A good system design strategy is to organize the program modules in such a method that are easy to develop and latter too, change. Structured design methods help developers to deal with the size and complexity of programs. Analysts generate instructions for the developers about how code should be composed and how pieces of code should fit together to form a program.

- To design a system, there are two possible approaches:
- Top-down Approach
- Bottom-up Approach

- **1. Top-down Approach:** This approach starts with the identification of the main components and then decomposing them into their more detailed sub-components.

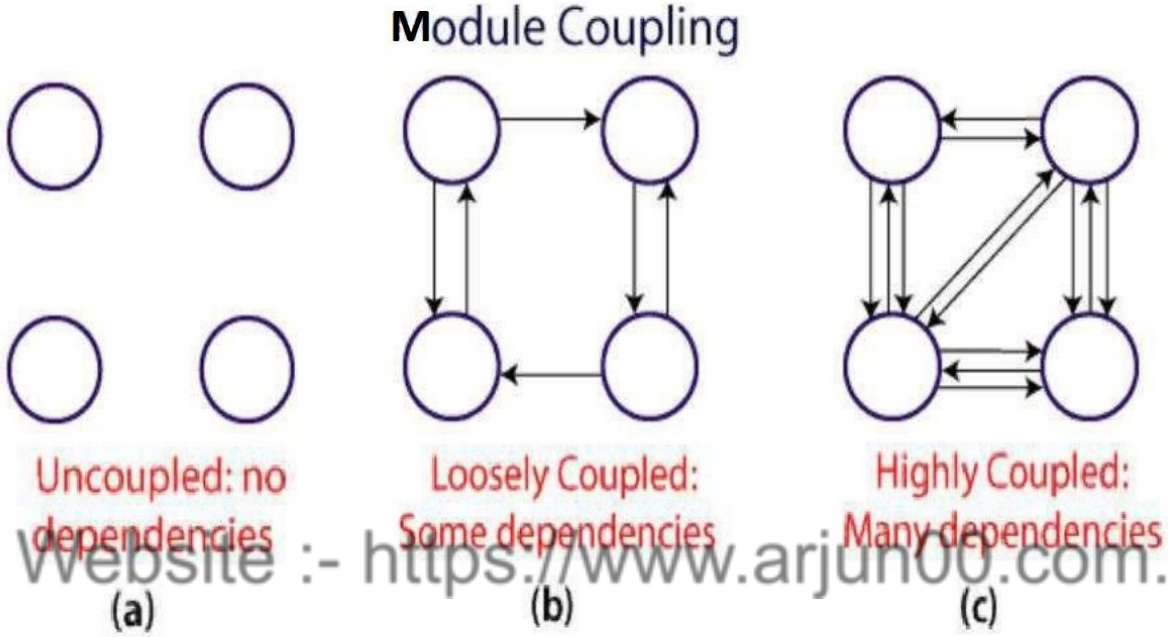


- **2. Bottom-up Approach:** A bottom-up approach begins with the lower details and moves towards up the hierarchy, as shown in fig. This approach is suitable in case of an existing system.

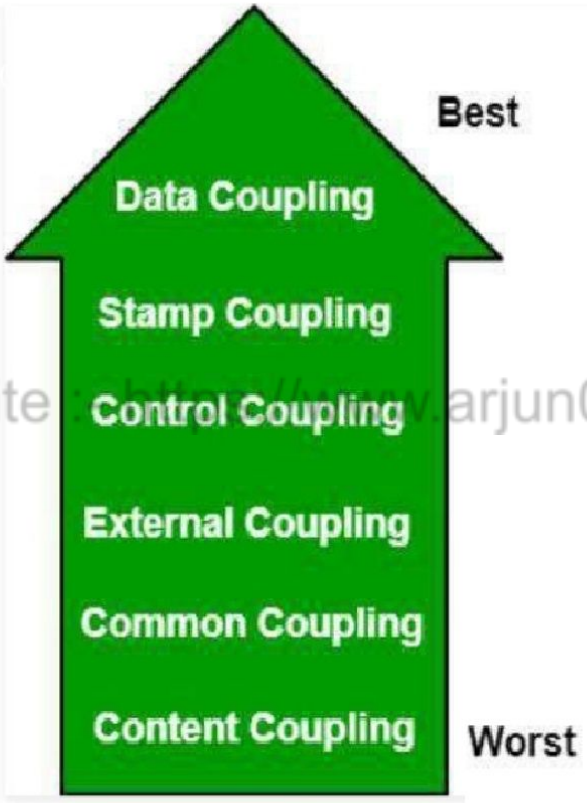


- Coupling and Cohesion
- Module Coupling
- In software engineering, the coupling is the degree of interdependence between software modules. Two modules that are tightly coupled are strongly dependent on each other. However, two modules that are loosely coupled are not dependent on each other. **Uncoupled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

The various types of coupling techniques are shown in fig:

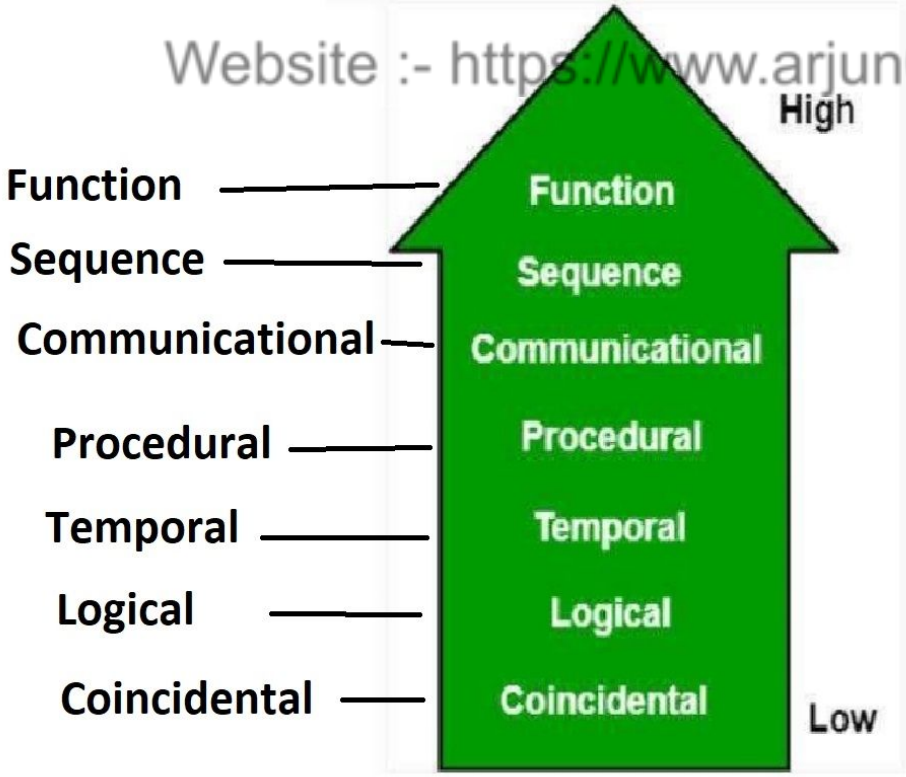


- A good design is the one that has low coupling. Coupling is measured by the number of relations between the modules. That is, the coupling increases as the number of calls between modules increase or the amount of shared data is large. Thus, it can be said that a design with high coupling will have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



- **Types of Coupling:**
- **Data Coupling:** If the dependency between the modules is based on the fact that they communicate by passing only data, then the modules are said to be data coupled. In data coupling, the components are independent of each other and communicate through data. Module communications don't contain tramp data. Example-customer billing system.
- **Stamp Coupling** In stamp coupling, the complete data structure is passed from one module to another module. Therefore, it involves tramp data. It may be necessary due to efficiency factors- this choice was made by the insightful designer, not a lazy programmer.

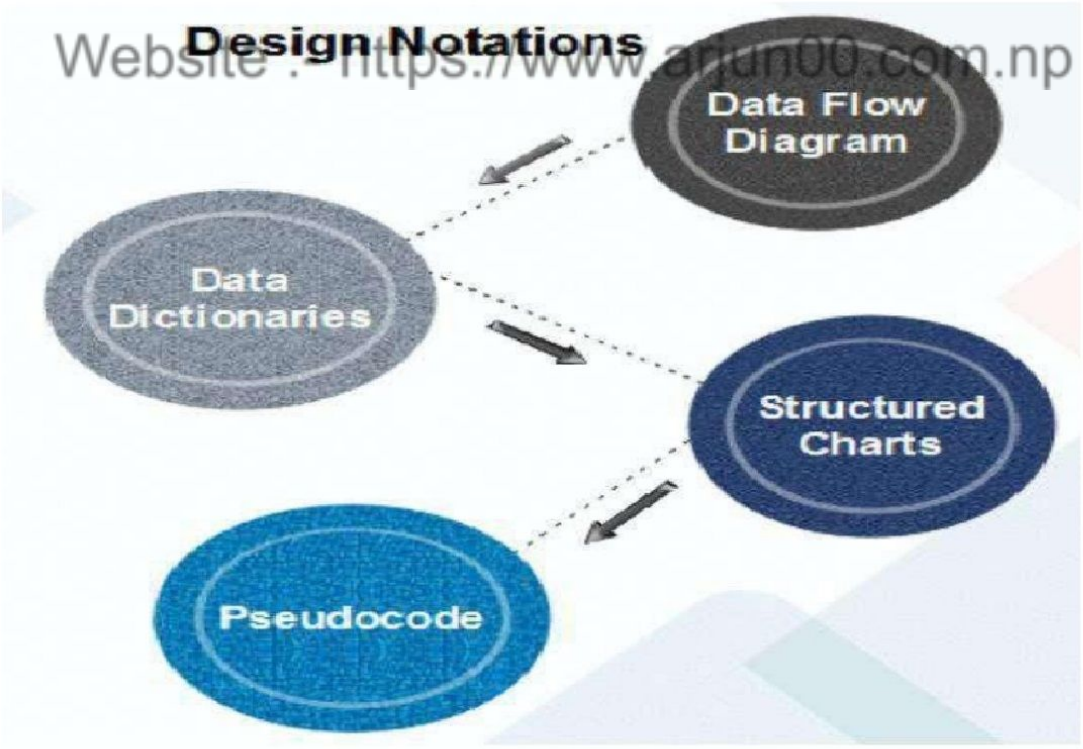
- **Control Coupling:** If the modules communicate by passing control information, then they are said to be control coupled. It can be bad if parameters indicate completely different behavior and good if parameters allow factoring and reuse of functionality. Example- sort function that takes comparison function as an argument.
- **External Coupling:** In external coupling, the modules depend on other modules, external to the software being developed or to a particular type of hardware. Ex- protocol, external file, device format, etc.
- **Common Coupling:** The modules have shared data such as global data structures. The changes in global data mean tracing back to all modules which access that data to evaluate the effect of the change. So it has got disadvantages like difficulty in reusing modules, reduced ability to control data accesses, and reduced maintainability.
- **Content Coupling:** In a content coupling, one module can modify the data of another module, or control flow is passed from one module to the other module. This is the worst form of coupling and should be avoided.
- **Cohesion:** Cohesion is a measure of the degree to which the elements of the module are functionally related. It is the degree to which all elements directed towards performing a single task are contained in the component. Basically, cohesion is the internal glue that keeps the module together. A good software design will have high cohesion.



- **Types of Cohesion:**
- **Functional Cohesion:** Every essential element for a single computation is contained in the component. A functional cohesion performs the task and functions. It is an ideal situation.
- **Sequential Cohesion:** An element outputs some data that becomes the input for other element, i.e., data flow between the parts. It occurs naturally in functional programming languages.
- **Communicational Cohesion:** Two elements operate on the same input data or contribute towards the same output data. Example- update record in the database and send it to the printer.
- **Procedural Cohesion:** Elements of procedural cohesion ensure the order of execution. Actions are still weakly connected and unlikely to be reusable. Ex- calculate student GPA, print student record, calculate cumulative GPA, print cumulative GPA.
- **Temporal Cohesion:** The elements are related by their timing involved. A module connected with temporal cohesion all the tasks must be executed in the same time span. This cohesion contains the code for initializing all the parts of the system. Lots of different activities occur, all at unit time.
- **Logical Cohesion:** The elements are logically related and not functionally. Ex- A component reads inputs from tape, disk, and network. All the code for these functions is in the same component. Operations are related, but the functions are significantly different.
- **Coincidental Cohesion:** The elements are not related(unrelated). The elements have no conceptual relationship other than location in source code. It is accidental and the worst form of cohesion. Ex- print next line and reverse the characters of a string in a single component.

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

- Function Oriented Design
- 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:

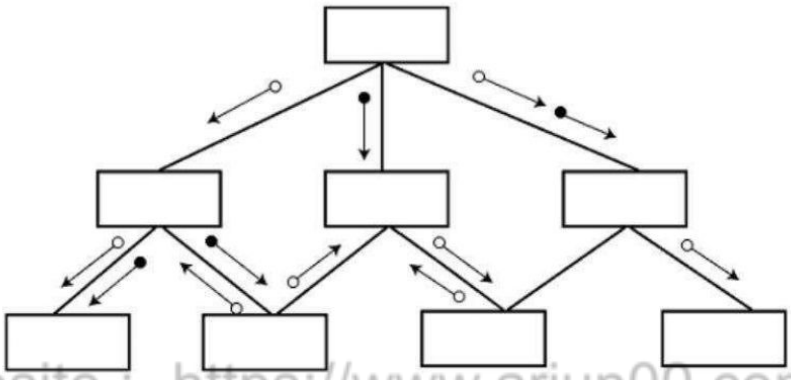


- Data Flow Diagram
- Data-flow design is concerned with designing a series of functional transformations that convert system inputs into the required outputs. The design is described as data-flow diagrams. These diagrams show how data flows through a system and how the output is derived from the input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

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.

- Data Dictionaries
- A data dictionary lists all data elements appearing in the DFD model of a system. The data items listed contain all data flows and the contents of all data stores looking on the DFDs in the DFD model of a system.
- Structured Charts
- It partitions a system into block boxes. A Black box system that functionality is known to the user without the knowledge of internal design.

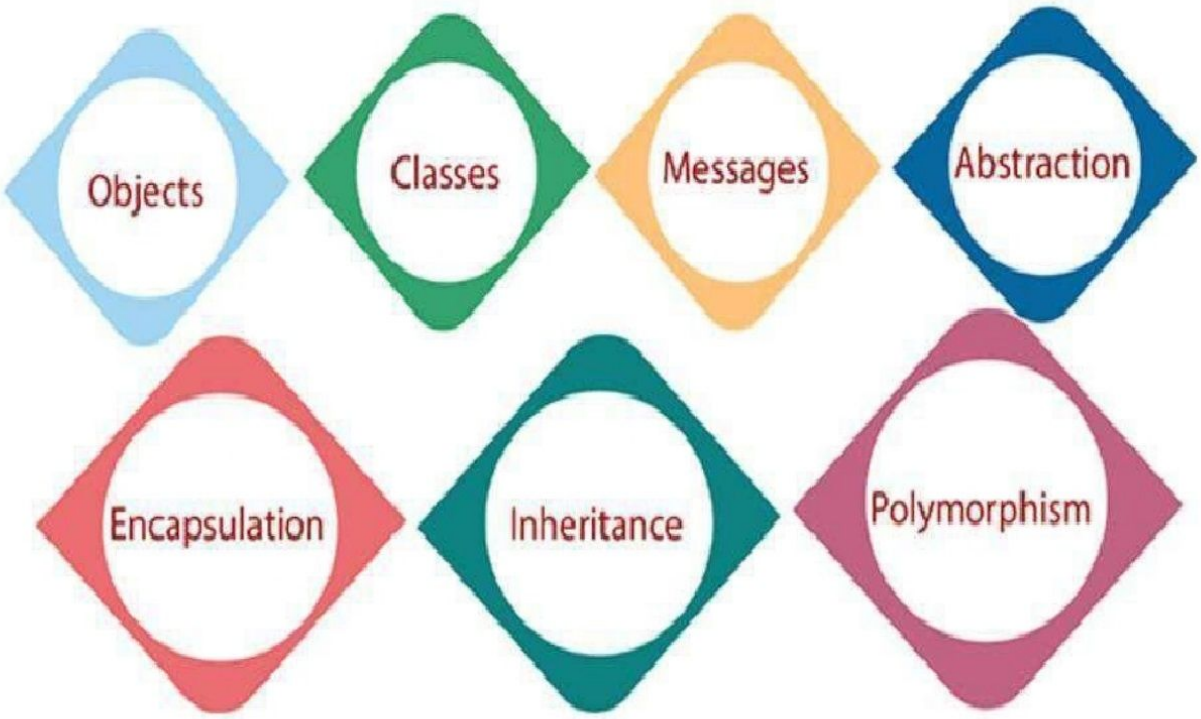


Hierarchical format of a structure chart

- Pseudo-code
- Pseudo-code notations can be used in both the preliminary and detailed design phases. Using pseudo-code, the designer describes system characteristics using short, concise, English Language phases that are structured by keywords such as If-Then-Else, While-Do, and End.
- Object-Oriented Design
- In the object-oriented design method, the system is viewed as a collection of objects (i.e., entities). The state is distributed among the objects, and each object handles its state data. For example, in a Library Automation Software, each library representative may be a separate object with its data and functions to operate on these data.

The different terms related to object design are:

Object Oriented Design



- **Objects:** All entities involved in the solution design are known as objects. For example, person, banks, company, and users are considered as objects. Every entity has some attributes associated with it and has some methods to perform on the attributes.
- **Classes:** A class is a generalized description of an object. An object is an instance of a class. A class defines all the attributes, which an object can have and methods, which represents the functionality of the object.
- **Messages:** Objects communicate by message passing. Messages consist of the integrity of the target object, the name of the requested operation, and any other action needed to perform the function. Messages are often implemented as procedure or function calls.
- **Abstraction** In object-oriented design, complexity is handled using abstraction. Abstraction is the removal of the irrelevant and the amplification of the essentials.
- **Encapsulation:** Encapsulation is also called an information hiding concept. The data and operations are linked to a single unit. Encapsulation not only bundles essential information of an object together but also restricts access to the data and methods from the outside world.
- **Inheritance:** OOD allows similar classes to stack up in a hierarchical manner where the lower or sub-classes can import, implement, and re-use allowed variables and functions from their immediate superclasses.This property of OOD is called an inheritance. This makes it easier to define a specific class and to create generalized classes from specific ones.
- **Polymorphism:** OOD languages provide a mechanism where methods performing similar tasks but vary in arguments, can be assigned the same name. This is known as polymorphism, which allows a single interface is performing functions for different types. Depending upon how the service is invoked, the respective portion of the code gets executed.

- **Correctness:**Software design should be correct as per requirement.
- **Completeness:**The design should have all components like data structures, modules, and external interfaces, etc.
- **Efficiency:**Resources should be used efficiently by the program.
- **Flexibility:**Able to modify on changing needs.

- **Consistency:**There should not be any inconsistency in the design.

- **Maintainability:** The design should be so simple so that it can be easily maintainable by other designers.
program.
- The software design process can be divided into the following three levels of phases of design:
- Interface Design
- Architectural Design
- Detailed Design
- **Interface Design:**
Interface design is the specification of the interaction between a system and its environment. this phase proceeds at a high level of abstraction with respect to the inner workings of the system i.e, during interface design, the internal of the systems are completely ignored and the system is treated as a black box.
- **Architectural Design:**
Architectural design is the specification of the major components of a system, their responsibilities, properties, interfaces, and the relationships and interactions between them. In architectural design, the overall structure of the system is chosen, but the internal details of major components are ignored.
- **Detailed Design:**
Design is the specification of the internal elements of all major system components, their properties, relationships, processing, and often their algorithms and the data structures.

COMPARISON FACTORS	FUNCTION ORIENTED DESIGN	OBJECT ORIENTED DESIGN
Abstraction	The basic abstractions, which are given to the user, are real world functions.	The basic abstractions are not the real world functions but are the data abstraction where the real world entities are represented.
Function	Functions are grouped together by which a higher level function is	Function are grouped together on the basis of the data they operate since
State information	In this approach the state information is often represented in a centralized shared memory.	In this approach the state information is not represented is not represented in a centralized memory but is implemented or distributed among the objects of the system.
Approach	It is a top down approach.	It is a bottom up approach.
Begins basis	Begins by considering the use case diagrams and	Begins by identifying objects and classes.
Decompose	In function oriented design we decompose in function/procedure level.	We decompose in class level.
Use	This approach is mainly used for computation sensitive application.	This approach is mainly used for evolving system which mimics a business or business case.

Software Metrics:

- 6.1 Software metrics: what & why?
- 6.2 Token count
- 6.3 Data structure metrics
- 6.4 Information flow metrics
- 6.5 Metrics analysis

6.1 SOFTWARE METRICS: WHAT AND WHY ?

Science begins with quantification; we cannot do physics without a notion of length and time; we cannot do thermodynamics until we measure temperature. All engineering disciplines have metrics (such as metrics for weight, density, wave length, pressure and temperature) to quantify various characteristics of their products. The most fundamental question we can ask is "how big is the program"? Without defining what big means, it is obvious that it makes no sense to say, "this program will need more testing than that program" unless we know "how big they are relative to one another. Comparing two strategies also needs a notion of size. The number of tests required by a strategy should be normalized to size. For example A needs 1.4 tests per unit of size, while strategy B needs 4.3 tests per unit of size.

What is meant by size was not obvious in the early phases of science development. Newton's use of mass instead of weight was a breakthrough for physics, and early researchers in thermodynamics had heat, temperature, and entropy hopelessly confused. Size is not obvious for the software. Metrics must be objective in the sense that the measurement process is algorithmic and will yield the same results no matter who applies it [BEIZ90]. To see what kinds of metrics, we need, let us ask some questions.

1. How to measure the size of a software?
2. How much will it cost to develop a software?
3. How many bugs can we expect?
4. When can we stop testing?
5. When can we release the software?
6. What is the complexity of a module?
7. What is the module strength and coupling?
8. What is the reliability at the time of release?
9. Which test technique is more effective?
10. Are we testing hard or are we testing smart?
11. Do we have a strong program or a weak test suite?

If we want an answer to the above questions, we will have to do our own measuring and fit our own empirical laws to the measured data. Most of the metrics are aimed at getting empirical laws that relate program size (however it be measured) to expected number of bugs, expected number of tests required to find bugs, test technique effectiveness, resource requirement, release instant, reliability and quality requirement, etc.

6.1.1 Definition

Software metrics can be defined as [GOOD93] *"The continuous application of measurement based techniques to the software development process and its products to supply meaningful and timely management information, together with the use of those techniques to improve that process and its products".*

6.1.4 Categories of Metrics

There are three categories of software metrics which are given below:

(*i*) **Product metrics:** describe the characteristics of the product such as size, complexity, design features, performance, efficiency, reliability, portability, etc.

(*ii*) **Process metrics:** describe the effectiveness and quality of the processes that produce the software product. Examples are:

- effort required in the process
- time to produce the product
- effectiveness of defect removal during development
- number of defects found during testing
- maturity of the process.

(*iii*) **Project metrics:** describe the project characteristics and execution. Examples are:

- number of software developers
- staffing pattern over the life cycle of the software
- cost and schedule
- productivity

Some metrics belong to multiple categories like quality metric may belong to all three categories. It focuses on the quality aspects of the product process, and the project. Some important metrics are discussed in subsequent sections of the chapter.

In the early 1970s, the late Professor Maurice Halstead and his co-workers at Purdue University developed the software science family of measures [HALS77]. Tokens are classified as either operators or operands. All software science measures are functions of the counts of these tokens.

Generally, any symbol or keyword in a program that specifies an algorithmic action is considered an operator, while a symbol used to represent data is considered an operand. Most punctuation marks are also categorized as operators. Variables, constants and even labels are operands. Operators consist of arithmetic symbols such as +, –, /, * and command names such as "while", "for", "printf", special symbols such as :=, braces, parentheses, and even function names such as "eof" (end of file). The size of the vocabulary of a program, which consists of the number of unique tokens used to build a program is defined as:

$$\eta = \eta_1 + \eta_2 \quad (6.1)$$

where η : vocabulary of a program

η_1 : number of unique operators

η_2 : number of unique operands

The length of the program in terms of the total number of tokens used is

$$N = N_1 + N_2 \quad (6.2)$$

where N : program length.

N_1 : total occurrences of operators

N_2 : total occurrences of operands

It should be noted that N is closely related to the lines of code (LOC) measure of program.

Data Structure Metrics

- Essentially the need for software development and other activities are to process data. Some data is input to a system, program or module; some data may be used internally, and some data is the output from a system, program, or module. Example:-

Program	Data Input	Internal Data	Data Output
Payroll	Name/Social Security No./Pay rate/Number of hours worked	Withholding rates Overtime Factors Insurance Premium Rates	Gross Pay withholding Net Pay Pay Ledgers
Spreadsheet	Item Names/Item Amounts/Relationships among Items	Cell computations Subtotal	Spreadsheet of items and totals
Software Planner	Program Size/No of Software developer on team	Model Parameter Constants Coefficients	Est. project effort Est. project duration

- That's why an important set of metrics which capture in the amount of data input, processed in an output form software. A count of this data structure is called Data Structured Metrics. In these concentrations is on variables (and given constant) within each module & ignores the input-output dependencies.
- There are some Data Structure metrics to computer the effort and time required to complete the project. There metrics are:

1. The Amount of Data.
2. The Usage of data within a Module.
3. Program weakness.
4. The sharing of Data among Modules.

- **1. The Amount of Data:** To measure the amount of Data, there are further many different metrics, and these are:
- **Number of variable (VARS):** In this metric, the Number of variables used in the program is counted.
- **Number of Operands (η_2):** In this metric, the Number of operands used in the program is counted. **η_2 = VARS + Constants + Labels**
- **Total number of occurrence of the variable (N2):** In this metric, the total number of occurrence of the variables are computed

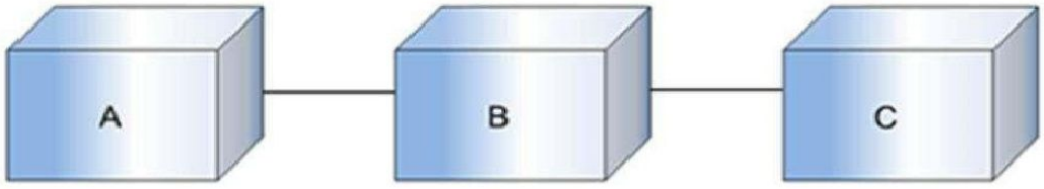
- **2. The Usage of data within a Module:** The measure this metric, the average numbers of live variables are computed. A variable is live from its first to its last references within the procedure.

$$\text{Average no of Live variables (LV)} = \frac{\text{Sum of count live variables}}{\text{Sum of count of executable statements}}$$

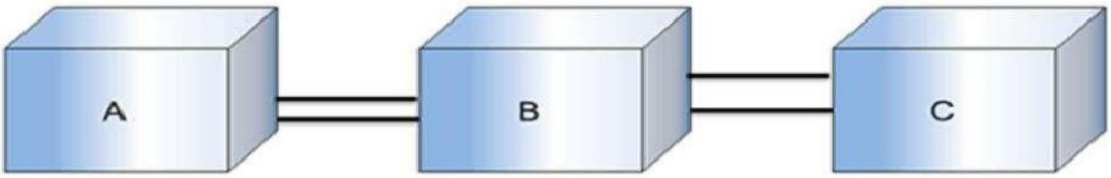
- **3. Program weakness:** Program weakness depends on its Modules weakness. If Modules are weak(less Cohesive), then it increases the effort and time metrics required to complete the project.

$$\text{Average life of variables } (\gamma) = \frac{\text{Sum of count live variables}}{\text{Sum of count of executable statements}}$$

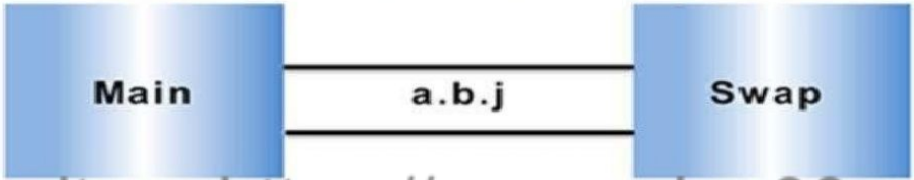
- **4.There Sharing of Data among Module:** As the data sharing between the Modules increases (higher Coupling), no parameter passing between Modules also increased, As a result, more effort and time are required to complete the project. So Sharing Data among Module is an important metrics to calculate effort and time.



Three modules from an imaginary program



"Pipes"of data shared among the modules



The data shared in program bubble

Information Flow Metrics

- The other set of metrics we would live to consider are known as Information Flow Metrics. The basis of information flow metrics is found upon the following concept the simplest system consists of the component, and it is the work that these components do and how they are fitted together that identify the complexity of the system. The following are the working definitions that are used in Information flow:
  - **Component:** Any element identified by decomposing a (software) system into it's constituent's parts.
  - **Cohesion:** The degree to which a component performs a single function.
  - **Coupling:** The term used to describe the degree of linkage between one component to others in the same system.
  - Information Flow metrics deal with this type of complexity by observing the flow of information among system components or modules. This metrics is given by **Henry and Kafura**. So it is also known as Henry and Kafura's Metric.
- This metrics is based on the measurement of the information flow among system modules. It is sensitive to the complexity due to interconnection among system component. This measure includes the complexity of a software module is defined to be the sum of complexities of the procedures included in the module. A process contributes complexity due to the following two factors.

- The complexity of the procedure code itself.
- The complexity due to the procedure's connections to its environment. The effect of the first factor has been included through LOC (Line Of Code) measure. For the quantification of the second factor, Henry and Kafura have defined two terms, namely FAN-IN and FAN-OUT.

- **FAN-IN:** FAN-IN of a procedure is the number of local flows into that procedure plus the number of data structures from which this procedure retrieve information.
- **FAN -OUT:** FAN-OUT is the number of local flows from that procedure plus the number of data structures which that procedure updates.
- Procedure Complexity = Length * (FAN-IN * FANOUT)**2

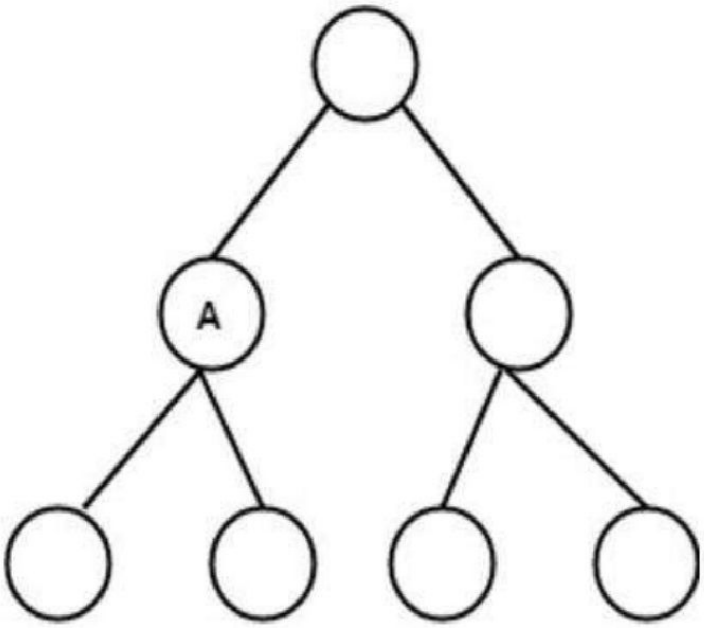


Fig: Aspects of Complexity

Case Tools For Software Metrics

- Many CASE tools (Computer Aided Software Engineering tools) exist for measuring software. They are either open source or are paid tools. Some of them are listed below:
- **Analyst4j tool** is based on the Eclipse platform and available as a stand-alone Rich Client Application or as an Eclipse IDE plug-in. It features search, metrics, analyzing quality, and report generation for Java programs.
- **CCCC is an open source command-line tool**. It analyzes C++ and Java lines and generates reports on various metrics, including Lines of Code and metrics proposed by Chidamber & Kemerer and Henry & Kafura.

- **Chidamber & Kemerer Java Metrics** is an open source command-line tool. It calculates the C&K object-oriented metrics by processing the byte-code of compiled Java.
- **Dependency Finder** is an open source. It is a suite of tools for analyzing compiled Java code. Its core is a dependency analysis application that extracts dependency graphs and mines them for useful information. This application comes as a command-line tool, a Swing-based application, and a web application.
- **Eclipse Metrics Plug-in 1.3.6** by Frank Sauer is an open source metrics calculation and dependency analyzer plugin for the Eclipse IDE. It measures various metrics and detects cycles in package and type dependencies.

- **Eclipse Metrics Plug-in 3.4** by Lance Walton is open source. It calculates various metrics during build cycles and warns, via the problems view, of metrics 'range violations'.
- **OOMeter** is an experimental software metrics tool developed by Alghamdi. It accepts Java/C# source code and UML models in XMI and calculates various metrics.
- **Semmle** is an Eclipse plug-in. It provides an SQL like querying language for object-oriented code, which allows searching for bugs, measure code metrics, etc.

Software Reliability:

- 7.1 Basic Concepts
- 7.2 Software quality
- 7.3 Software reliability model
- 7.4 Capability maturity model (CMM)

Software Reliability

Basic Concepts

There are three phases in the life of any hardware component i.e., burn-in, useful life & wear-out.

In **burn-in phase**, failure rate is quite high initially, and it starts decreasing gradually as the time progresses.

During **useful life period**, failure rate is approximately constant.

Failure rate increase in **wear-out phase** due to wearing out/aging of components. The best period is useful life period. The shape of this curve is like a "bath tub" and that is why it is known as bath tub curve. The "bath tub curve" is given in Fig. 7.1.

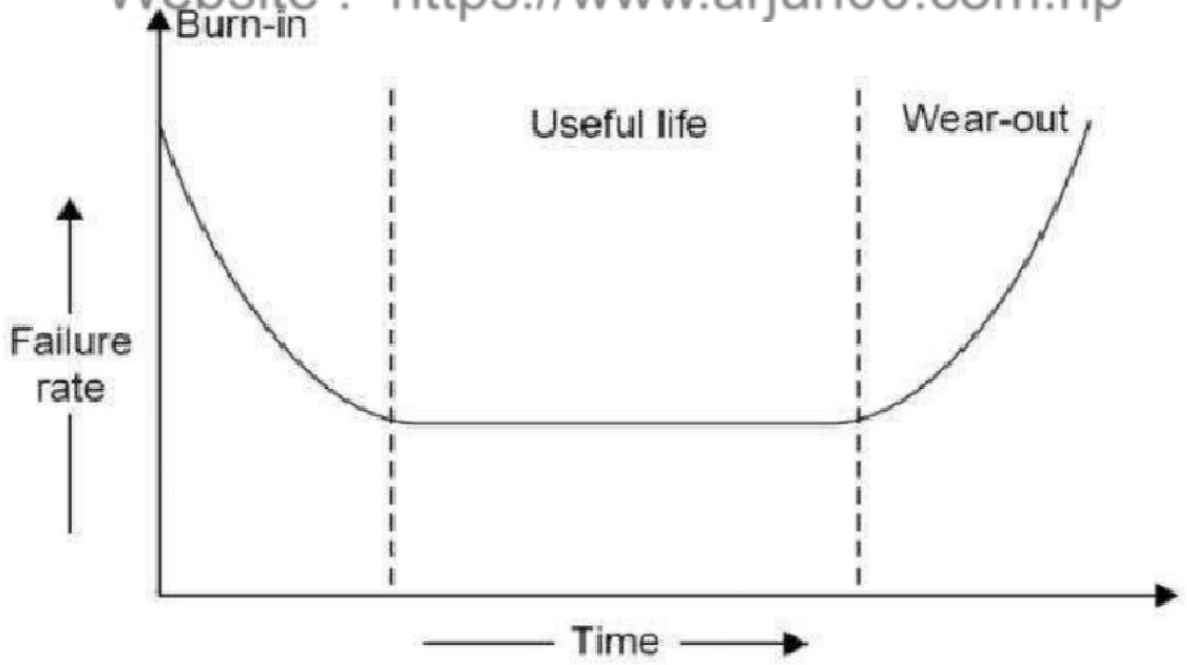


Fig. 7.1: Bath tub curve of hardware reliability.

We do not have wear out phase in software. The expected curve for software is given in fig. 7.2.

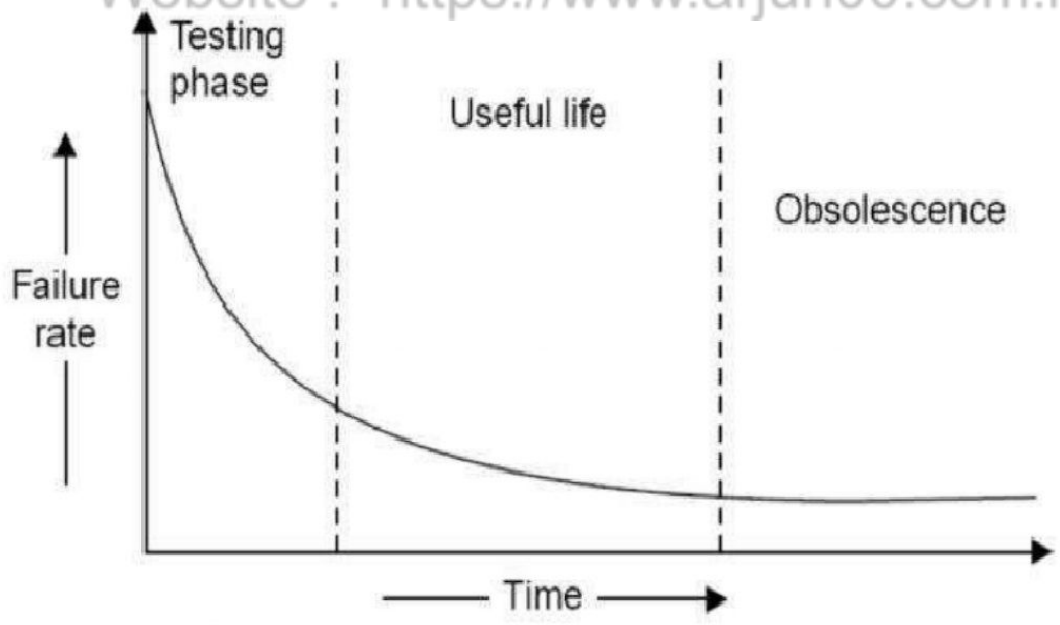


Fig. 7.2: Software reliability curve (failure rate versus time)

Software may be retired only if it becomes obsolete. Some of contributing factors are given below:

- ✓ change in environment
- ✓ change in infrastructure/technology
- ✓ major change in requirements
- ✓ increase in complexity
- ✓ extremely difficult to maintain
- ✓ deterioration in structure of the code
- ✓ slow execution speed
- ✓ poor graphical user interfaces

What is Software Reliability?

"Software reliability means operational reliability. Who cares how many bugs are in the program?

As per IEEE standard: "Software reliability is defined as the ability of a system or component to perform its required functions under stated conditions for a specified period of time".

Software reliability is also defined as the probability that a software system fulfills its assigned task in a given environment for a predefined number of input cases, assuming that the hardware and the inputs are free of error.

"It is the probability of a failure free operation of a program for a specified time in a specified environment".

▪ Failures and Faults

A fault is the defect in the program that, when executed under particular conditions, causes a failure.

The execution time for a program is the time that is actually spent by a processor in executing the instructions of that program. The second kind of time is calendar time. It is the familiar time that we normally experience.

There are four general ways of characterising failure occurrences in time:

1. time of failure,
2. time interval between failures,
3. cumulative failure experienced up to a given time,
4. failures experienced in a time interval.

Failure Number	Failure Time (sec)	Failure interval (sec)
1	8	8
2	18	10
3	25	7
4	36	11
5	45	9
6	57	12
7	71	14
8	86	15
9	104	18
10	124	20
11	143	19
12	169	26
13	197	28
14	222	25
15	250	28

Table 7.1: Time based failure specification

Time (sec)	Cumulative Failures	Failure in interval (30 sec)
30	3	3
60	6	3
90	8	2
120	9	1
150	11	2
180	12	1
210	13	1
240	14	1

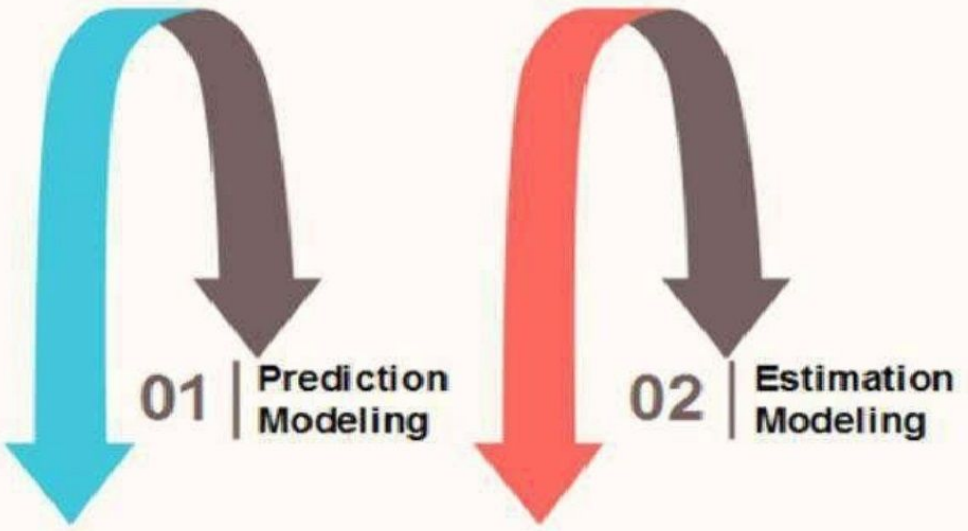
Table 7.2: Failure based failure specification

Software Reliability Models

- A software reliability model indicates the form of a random process that defines the behavior of software failures to time.
- Most software models contain the following parts:
- Assumptions
- Factors

Software Reliability Modeling Techniques

Software modeling techniques can be divided into two sub-categories:

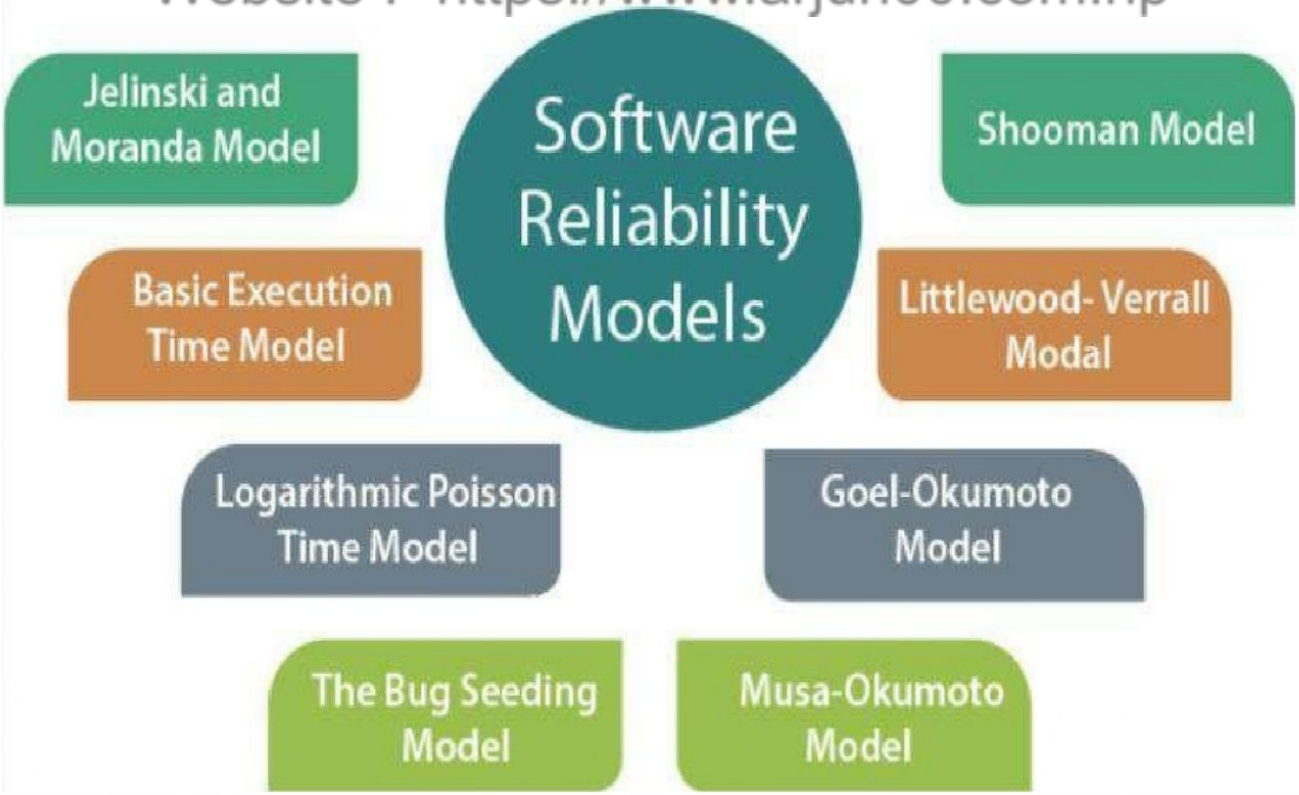


Differentiate between software reliability prediction models and software reliability estimation models

Basics	Prediction Models	Estimation Models
Data Reference	Uses historical information	Uses data from the current software development effort.
When used in development cycle	Usually made before development or test phases; can be used as early as concept phase.	Usually made later in the life cycle (after some data have been collected); not typically used in concept or development phases.
Time Frame	Predict reliability at some future time.	Estimate reliability at either present or some next time.

- Reliability Models
- A reliability growth model is a numerical model of software reliability, which predicts how software reliability should improve over time as errors are discovered and repaired. These models help the manager in deciding how much efforts should be devoted to testing. The objective of the project manager is to test and debug the system until the required level of reliability is reached.

Following are the Software Reliability Models are:



SOFTWARE RELIABILITY MODELS

- Jelinski and Moranda Model
  - Realizes each time an error is repaired reliability does not increase by a constant amount.
  - Reliability improvement due to fixing of an error is assumed to be proportional to the number of errors present in the system at that time.

- Littlewood and Verall's Model
  - Assumes different fault have different sizes, thereby contributing unequally to failures.
  - Large sized faults tends to be detected and fixed earlier
  - As number of errors is driven down with the progress in test, so is the average error size, causing a law of diminishing return in debugging

MUSA'S MODEL

- Assumptions:-
  - Faults are independent and distributed with constant rate of encounter.
  - Well mixed types of instructions, execution time between failures is large compared to instruction execution time.
  - Set of inputs for each run selected randomly.
  - All failures are observed, implied by definition.
  - Fault causing failure is corrected immediately, otherwise reoccurrence of that failure is not counted.

- Basic Execution Time Model
- This model was established by J.D. Musa in 1979, and it is based on execution time. The basic execution model is the most popular and generally used reliability growth model, mainly because:
- It is practical, simple, and easy to understand.
- Its parameters clearly relate to the physical world.
- It can be used for accurate reliability prediction.
- The basic execution model determines failure behavior initially using execution time. Execution time may later be converted in calendar time.
- The failure behavior is a **nonhomogeneous Poisson process**, which means the associated probability distribution is a Poisson
- process whose characteristics vary in time.
- Prime Ministers of India | List of Prime Minister of India (1947-2020)

Goel-Okumoto Model

- The Goel-Okumoto model (also called as exponential NHPP model) is based on the following assumptions:
- All faults in a program are mutually independent of the failure detection point of view.
- The number of failures detected at any time is proportional to the current number of faults in a program. This means that the probability of the failures for faults actually occurring, i.e., detected, is constant.

Software Qulaity

- Software Quality
- Software quality product is defined in term of its fitness of purpose. That is, a quality product does precisely what the users want it to do. For software products, the fitness of use is generally explained in terms of satisfaction of the requirements laid down in the SRS document.
- **The modern view of a quality associated with a software product several quality methods such as the following:**
- **Portability:** A software device is said to be portable, if it can be freely made to work in various operating system environments, in multiple machines, with other software products, etc.
- **Usability:** A software product has better usability if various categories of users can easily invoke the functions of the product.

- **Reusability:** A software product has excellent reusability if different modules of the product can quickly be reused to develop new products.
- **Correctness:** A software product is correct if various requirements as specified in the SRS document have been correctly implemented.
- **Maintainability:** A software product is maintainable if bugs can be easily corrected as and when they show up, new tasks can be easily added to the product, and the functionalities of the product can be easily modified, etc.

Capability maturity model (CMM)

- CMM was developed by the Software Engineering Institute (SEI) at Carnegie Mellon University in 1987.
- It is not a software process model. It is a framework that is used to analyze the approach and techniques followed by any organization to develop software products.
- It also provides guidelines to further enhance the maturity of the process used to develop those software products.
- It is based on profound feedback and development practices adopted by the most successful organizations worldwide.

- This model describes a strategy for software process improvement that should be followed by moving through 5 different levels.
- Each level of maturity shows a process capability level. All the levels except level-1 are further described by Key Process Areas (KPA's).

- **Key Process Areas (KPA's):**
  Each of these KPA's defines the basic requirements that should be met by a software process in order to satisfy the KPA and achieve that level of maturity.

- The 5 levels of CMM are as follows:
- **Level-1: Initial –**
- No KPA's defined.
- Processes followed are Adhoc and immature and are not well defined.
- Unstable environment for software development.
- No basis for predicting product quality, time for completion, etc.

Level-2: Repeatable –

- Focuses on establishing basic project management policies.
- Experience with earlier projects is used for managing new similar natured projects.
- **Project Planning-** It includes defining resources required, goals, constraints, etc. for the project. It presents a detailed plan to be followed systematically for the successful completion of good quality software.
- **Configuration Management-** The focus is on maintaining the performance of the software product, including all its components, for the entire lifecycle.
- **Requirements Management-** It includes the management of customer reviews and feedback which result in some changes in the requirement set. It also consists of accommodation of those modified requirements.
- **Subcontract Management-** It focuses on the effective management of qualified software contractors i.e. it manages the parts of the software which are developed by third parties.

- **Software Quality Assurance-** It guarantees a good quality software product by following certain rules and quality standard guidelines while developing.
- **Level-3: Defined –**
- At this level, documentation of the standard guidelines and procedures takes place.
- It is a well-defined integrated set of project-specific software engineering and management processes.
- **Peer Reviews-** In this method, defects are removed by using a number of review methods like walkthroughs, inspections, buddy checks, etc.
- **Intergroup Coordination-** It consists of planned interactions between different development teams to ensure efficient and proper fulfillment of customer needs.
- **Organization Process Definition-** Its key focus is on the development and maintenance of the standard development processes.
- **Organization Process Focus-** It includes activities and practices that should be followed to improve the process capabilities of an organization.
- **Training Programs-** It focuses on the enhancement of knowledge and skills of the team members including the developers and ensuring an increase in work efficiency.

- **Level-4: Managed –**
- At this stage, quantitative quality goals are set for the organization for software products as well as software processes.
- The measurements made help the organization to predict the product and process quality within some limits defined quantitatively.
- **Software Quality Management-** It includes the establishment of plans and strategies to develop quantitative analysis and understanding of the product's quality.
- **Quantitative Management-** It focuses on controlling the project performance in a quantitative manner.

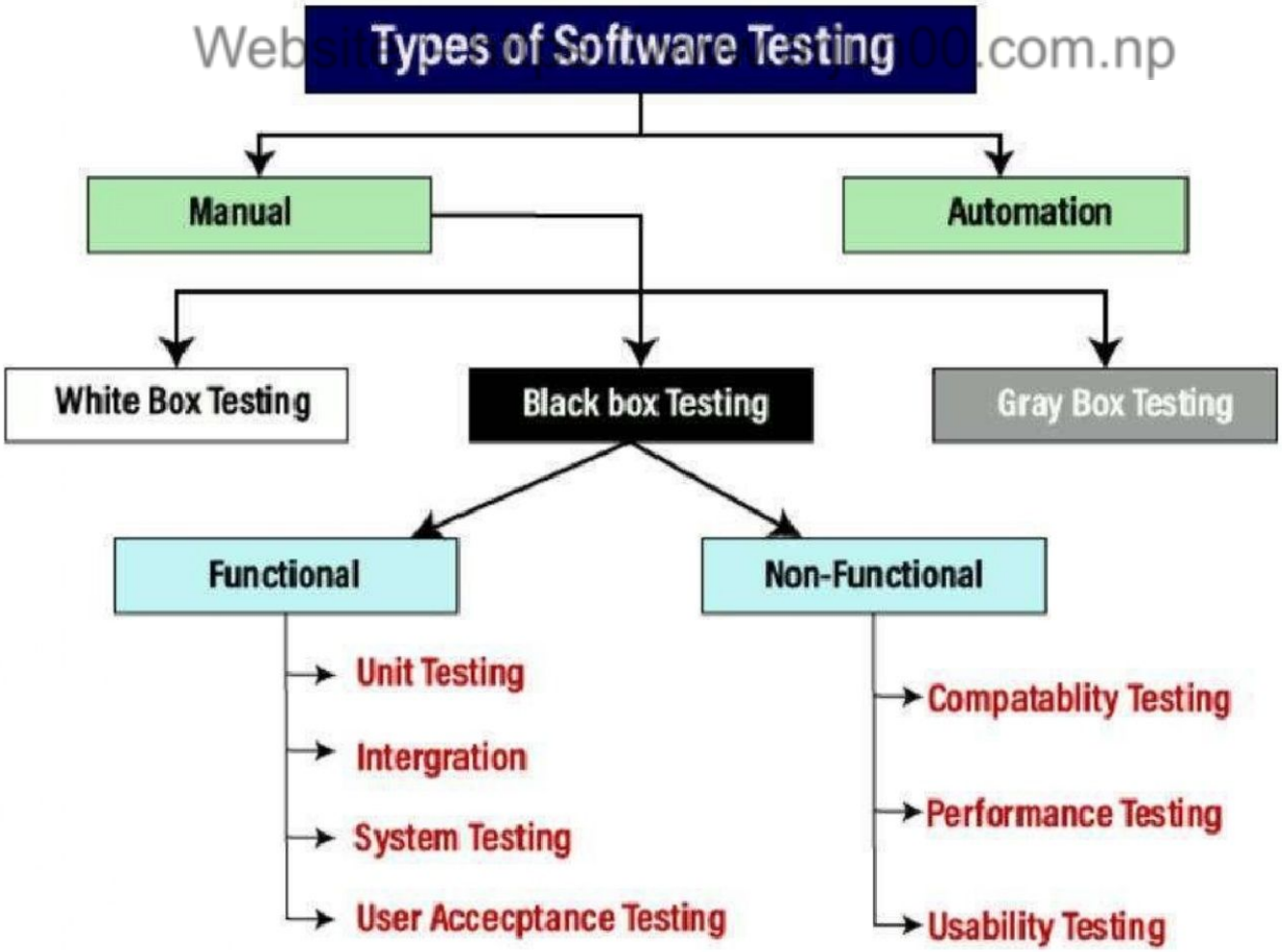
- **Level-5: Optimizing –**
- This is the highest level of process maturity in CMM and focuses on continuous process improvement in the organization using quantitative feedback.
- Use of new tools, techniques, and evaluation of software processes is done to prevent recurrence of known defects.
- **Process Change Management-** Its focus is on the continuous improvement of the organization's software processes to improve productivity, quality, and cycle time for the software product.
- **Technology Change Management-** It consists of the identification and use of new technologies to improve product quality and decrease product development time.

- **Defect Prevention-** It focuses on the identification of causes of defects and prevents them from recurring in future projects by improving project-defined processes.

Software Testing:

- 8.1 Testing process
- 8.2 Some important terminologies
- 8.3 Unit testing
- 8.4 Integration testing
- 8.5 System testing
- 8.6 Regression Testing
- 8.7 Performance testing
- 8.8 White Box testing and black box testing
- 8.9 Acceptance testing
- 8.10 Alpha and Beta testing
- 8.11 Debugging techniques, tools and approaches

- **Introduction:-**
- Testing is the process of executing a program with the aim of finding errors. To make our software perform well it should be error-free. If testing is done successfully it will remove all the errors from the software.



Types of Testing:-

- **1. Unit Testing**
- It focuses on the smallest unit of software design. In this, we test an individual unit or group of interrelated units. It is often done by the programmer by using sample input and observing its corresponding outputs.
  Example:
- a) In a program we are checking if loop, method or function is working fine
- b) Misunderstood or incorrect, arithmetic precedence.
- c) Incorrect initialization

- **2. Integration Testing**
- The objective is to take unit tested components and build a program structure that has been dictated by design. Integration testing is testing in which a group of components is combined to produce output.
- Integration testing is of four types: (i) Top-down (ii) Bottom-up (iii) Sandwich (iv) Big-Bang
  Example
- a) Black Box testing:- It is used for validation. In this we ignore internal working mechanism and focuse on **what is the output?**.
- (b) White Box testing:- It is used for verification. In this we focus on internal mechanism i.e. **how the output is achieved?**

- **White Box Testing:**

- It is also called Glass Box, Clear Box, Structural Testing. White Box Testing is based on the application's internal code structure. In white-box testing, an internal perspective of the system, as well as programming skills, are used to design test cases. This testing is usually done at the unit level.

- **System Testing**
- This software is tested such that it works fine for the different operating systems. It is covered under the black box testing technique. In this, we just focus on the required input and output without focusing on internal working.
  In this, we have security testing, recovery testing, stress testing, and performance testing
  Example:

- This include functional as well as non functional testing
- a) Black Box testing:- It is used for validation. In this we ignore internal working mechanism and focuse on **what is the output?**.
- (b) White Box testing:- It is used for verification. In this we focus on internal mechanism i.e. **how the output is achieved?**

- **White Box Testing:**
- It is also called Glass Box, Clear Box, Structural Testing. White Box Testing is based on the application's internal code structure. In white-box testing, an internal perspective of the system, as well as programming skills, are used to design test cases. This testing is usually done at the unit level.

- **System Testing**
- This software is tested such that it works fine for the different operating systems. It is covered under the black box testing technique. In this, we just focus on the required input and output without focusing on internal working.
  In this, we have security testing, recovery testing, stress testing, and performance testing
  Example:
- This include functional as well as non functional testing

- **3. Regression Testing**
- Every time a new module is added leads to changes in the program. This type of testing makes sure that the whole component works properly even after adding components to the complete program.
  Example
- In school record suppose we have module staff, students and finance combining these modules and checking if on integration these module works fine is regression testing
- Acceptance Testing
- When the software is ready to hand over to the customer it has to go through last phase of testing where it is tested for user-interaction and response. This is important because even if the software matches all user requirements and if user does not like the way it appears or works, it may be rejected.
- Types of Acceptance Testing:
- **5. Alpha Testing**
- This is a type of validation testing. It is a type of *acceptance testing* which is done before the product is released to customers. It is typically done by QA people.
  Example:
- When software testing is performed internally within the organization

- **6. Beta Testing**
- The beta test is conducted at one or more customer sites by the end-user of the software. This version is released for a limited number of users for testing in a real-time environment
  Example:
- When software testing is performed for the limited number of people

- **Performance Testing**
- It is designed to test the run-time performance of software within the context of an integrated system. It is used to test the speed and effectiveness of the program. It is also called load testing. In it we check, what is the performance of the system in the given load.
  Example:
- Checking number of processor cycles.

Debugging :

- Debugging is the process of finding and resolving defects or problems within a computer program that prevent correct operation of computer software or a system.

- **Need for debugging:**
  Once errors are known during a program code, it's necessary to initial establish the precise program statements liable for the errors and so to repair them.

Challenges in Debugging:

- There are lot of problems at the same time as acting the debugging. These are the following:
- Debugging is finished through the individual that evolved the software program and it's miles difficult for that person to acknowledge that an error was made.
- Debugging is typically performed under a tremendous amount of pressure to fix the supported error as quick as possible.
- It can be difficult to accurately reproduce input conditions.
- Compared to the alternative software program improvement activities, relatively little research, literature and formal preparation exist at the procedure of debugging.

- **Debugging Approaches:**

The following are a number of approaches popularly adopted by programmers for debugging.

- **Brute Force Method:**

This is the foremost common technique of debugging however is that the least economical method. during this approach, the program is loaded with print statements to print the intermediate values with the hope that a number of the written values can facilitate to spot the statement in error. This approach becomes a lot of systematic with the utilisation of a symbolic program (also known as a source code debugger), as a result of values of various variables will be simply checked and breakpoints and watch-points can be easily set to check the values of variables effortlessly.

- **Backtracking:**
  This is additionally a reasonably common approach. during this approach, starting from the statement at which an error symptom has been discovered, the source code is derived backward till the error is discovered. sadly, because the variety of supply lines to be derived back will increase, the quantity of potential backward methods will increase and should become unimaginably large so limiting the utilisation of this approach.
- **Cause Elimination Method:**
  In this approach, a listing of causes that may presumably have contributed to the error symptom is developed and tests are conducted to eliminate every error. A connected technique of identification of the error from the error symptom is that the package fault tree analysis.
- **Program Slicing:**
  This technique is analogous to backtracking. Here the search house is reduced by process slices. A slice of a program for a specific variable at a particular statement is that the set of supply lines preceding this statement which will influence the worth of that variable

- **Debugging Tools:**
  Debugging tool is a computer program that is used to test and debug other programs. A lot of public domain software like gdb and dbx are available for debugging. They offer console-based command line interfaces. Examples of automated debugging tools include code based tracers, profilers, interpreters, etc.
  Some of the widely used debuggers are:
- Radare2
- WinDbg
- Valgrind

Software Maintenance:

- 9.1 Need for software maintenance
- 9.2 Types of software maintenance
- 9.3 Software maintenance process model.
- 9.4 Software maintenance cost

Software Maintenance

- Software Maintenance is the process of modifying a software product after it has been delivered to the customer. The main purpose of software maintenance is to modify and update software applications after delivery to correct faults and to improve performance.

Need for Maintenance –

Software Maintenance must be performed in order to:

- Correct faults.
- Improve the design.
- Implement enhancements.
- Interface with other systems.
- Accommodate programs so that different hardware, software, system features, and telecommunications facilities can be used.
- Migrate legacy software.
- Retire software.

- **Categories of Software Maintenance/Types –**
  Maintenance can be divided into the following:

- **Corrective maintenance:**
  Corrective maintenance of a software product may be essential either to rectify some bugs observed while the system is in use, or to enhance the performance of the system.
  Adaptive maintenance:
  This includes modifications and updations when the customers need the product to run on new platforms, on new operating systems, or when they need the product to interface with new hardware and software.

- **Perfective maintenance:**
  A software product needs maintenance to support the new features that the users want or to change different types of functionalities of the system according to the customer demands.

- **Preventive maintenance:**
  This type of maintenance includes modifications and updations to prevent future problems of the software. It goals to attend problems, which are not significant at this moment but may cause serious issues in future.

Process of Software Maintenance:

- Software Maintenance is an important phase of Software Development Life Cycle (SDLC), and it is implemented in the system through a proper software maintenance process, known as **Software Maintenance Life Cycle (SMLC)**. This life cycle consists of seven different phases, each of which can be used in iterative manner and can be extended so that customized items and processes can be included. These seven phases of Software Maintenance process are:

- **Identification Phase:**
- In this phase, the requests for modifications in the software are identified and analysed. Each of the requested modification is then assessed to determine and classify the type of maintenance activity it requires. This is either generated by the system itself, via logs or error messages, or by the user.

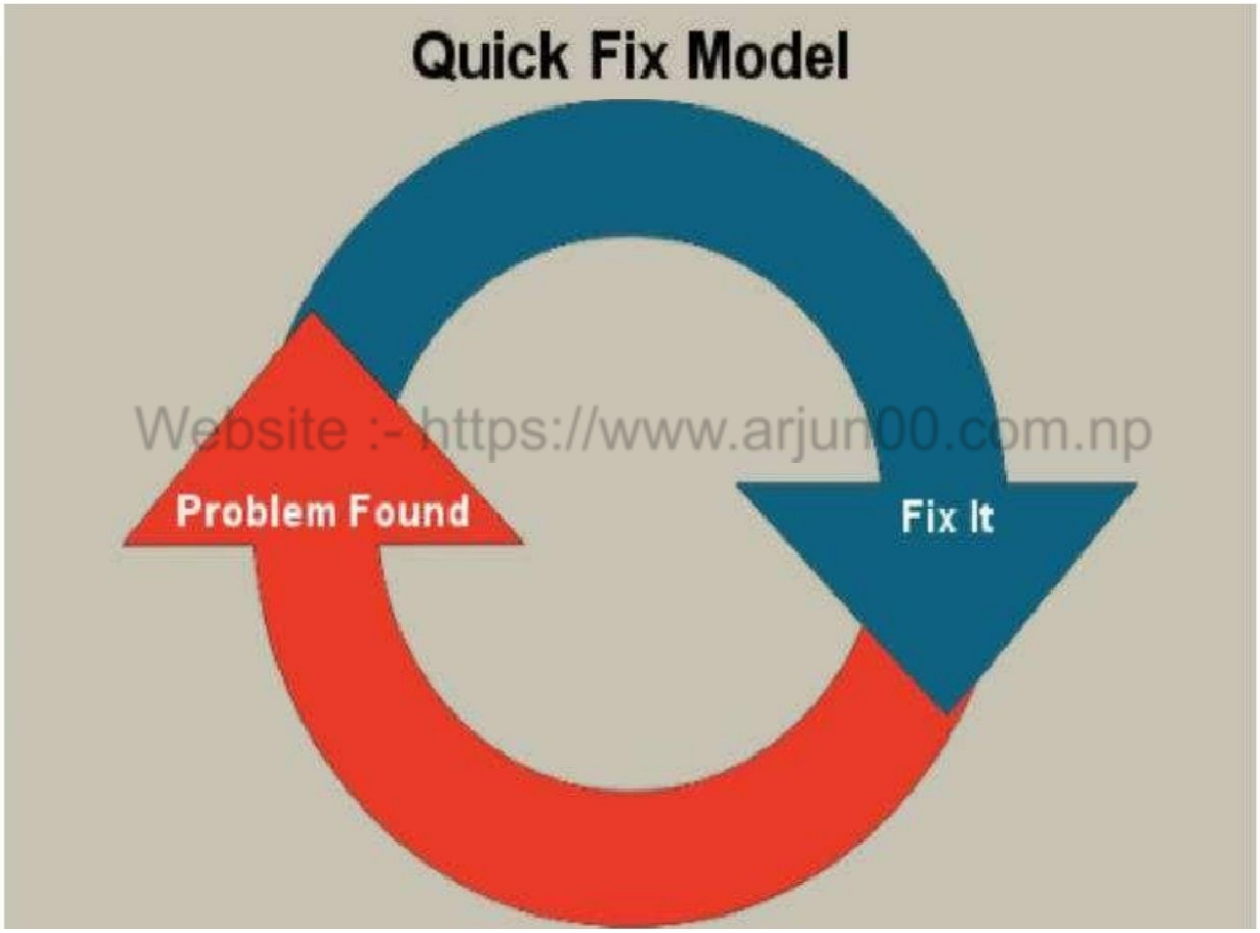
- **Analysis Phase:**
- The feasibility and scope of each validated modification request are determined and a plan is prepared to incorporate the changes in the software. The input attribute comprises validated modification request, initial estimate of resources, project documentation, and repository information. The cost of modification and maintenance is also estimated.
- **Design Phase:**
- The new modules that need to be replaced or modified are designed as per the requirements specified in the earlier stages. Test cases are developed for the new design including the safety and security issues. These test cases are created for the **validation and verification** of the system.
- **Implementation Phase:**
- In the implementation phase, the actual modification in the software code are made, new features that support the specifications of the present software are added, and the modified software is installed. The new modules are coded with the assistance of structured design created in the design phase.
- **System Testing Phase:**
- **Regression testing** is performed on the modified system to ensure that no defect, error or bug is left undetected. Furthermore, it validates that no new faults are introduced in the software as a result of maintenance activity. **Integration testing** is also carried out between new modules and the system.

- **Acceptance Testing Phase:**
- **Acceptance testing** is performed on the fully integrated system by the user or by the third party specified by the end user. The main objective of this testing is to verify that all the features of the software are according to the requirements stated in the modification request.
- **Delivery Phase:**
- Once the acceptance testing is successfully accomplished, the modified system is delivered to the users. In addition to this, the user is provided proper consisting of manuals and help files that describe the operation of the software along with its hardware specifications. The final testing of the system is done by the client after the system is delivered.

Software Maintenance Models:

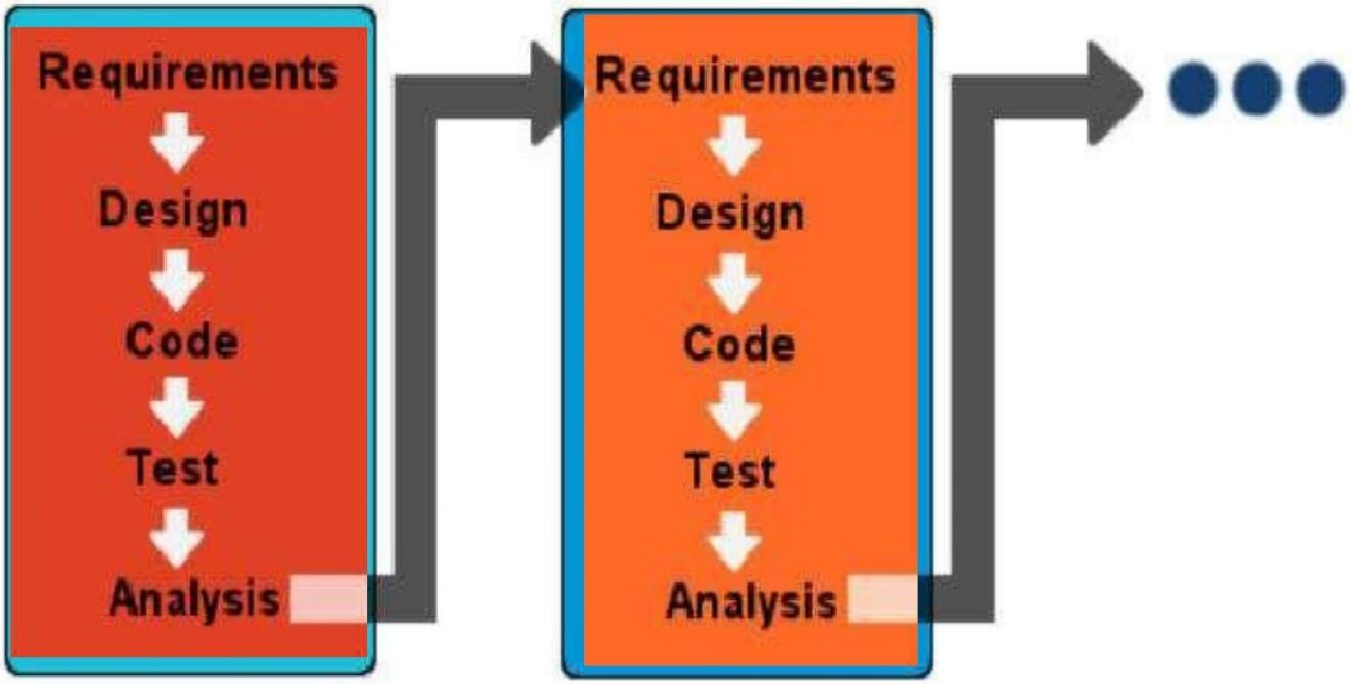
- To overcome internal as well as external problems of the software, Software maintenance models are proposed. These models use different approaches and techniques to simplify the process of maintenance as well as to make is cost effective. Software maintenance models that are of most importance are:

- Quick-Fix Model:
- This is an ad hoc approach used for maintaining the software system. The objective of this model is to identify the problem and then fix it as quickly as possible. The advantage is that it performs its work quickly and at a low cost. This model is an approach to modify the software code with little consideration for its impact on the overall structure of the software system.



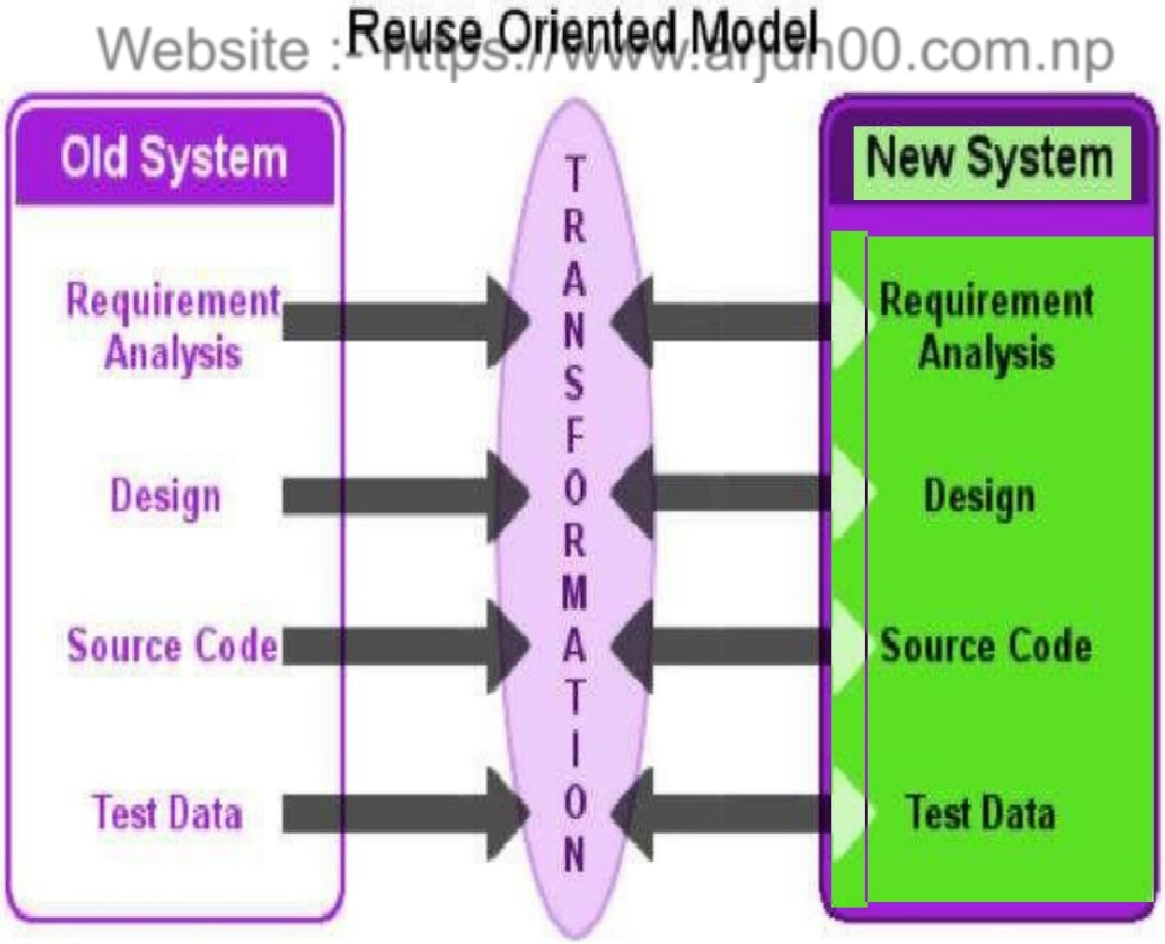
- Iterative Enhancement Model:
- Iterative enhancement model considers the changes made to the system are iterative in nature. This model incorporates changes in the software based on the analysis of the existing system. It assumes complete documentation of the software is available in the beginning. Moreover, it attempts to control complexity and tries to maintain good design.

Iterative Enhancement Model

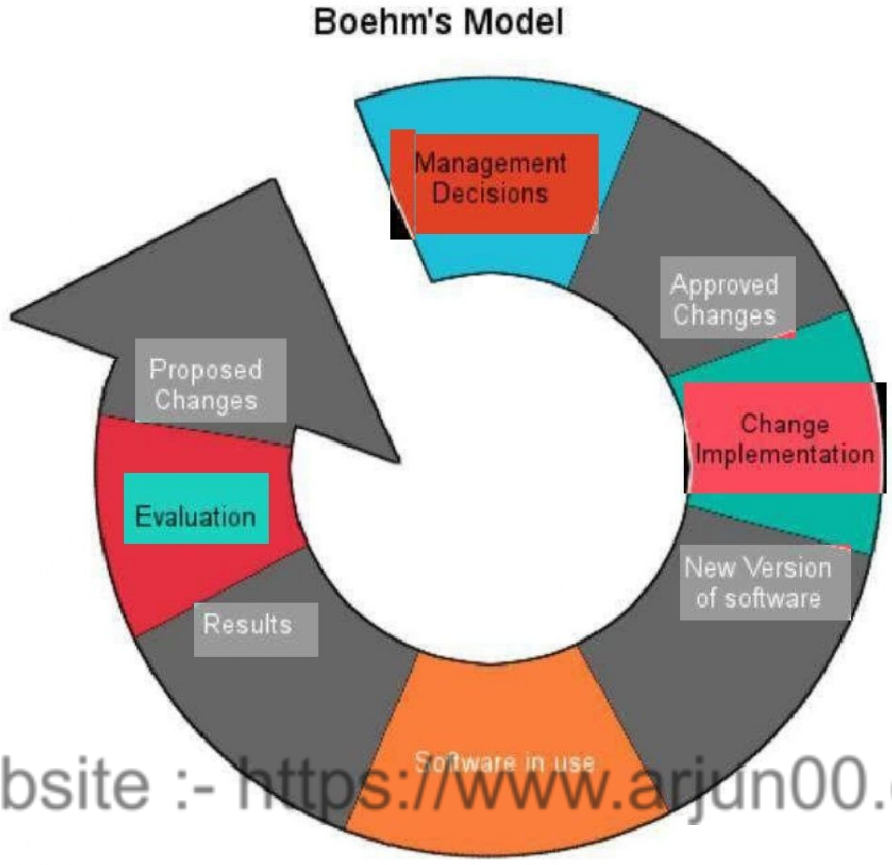


- Iterative Enhancement Model is divided into three stages:

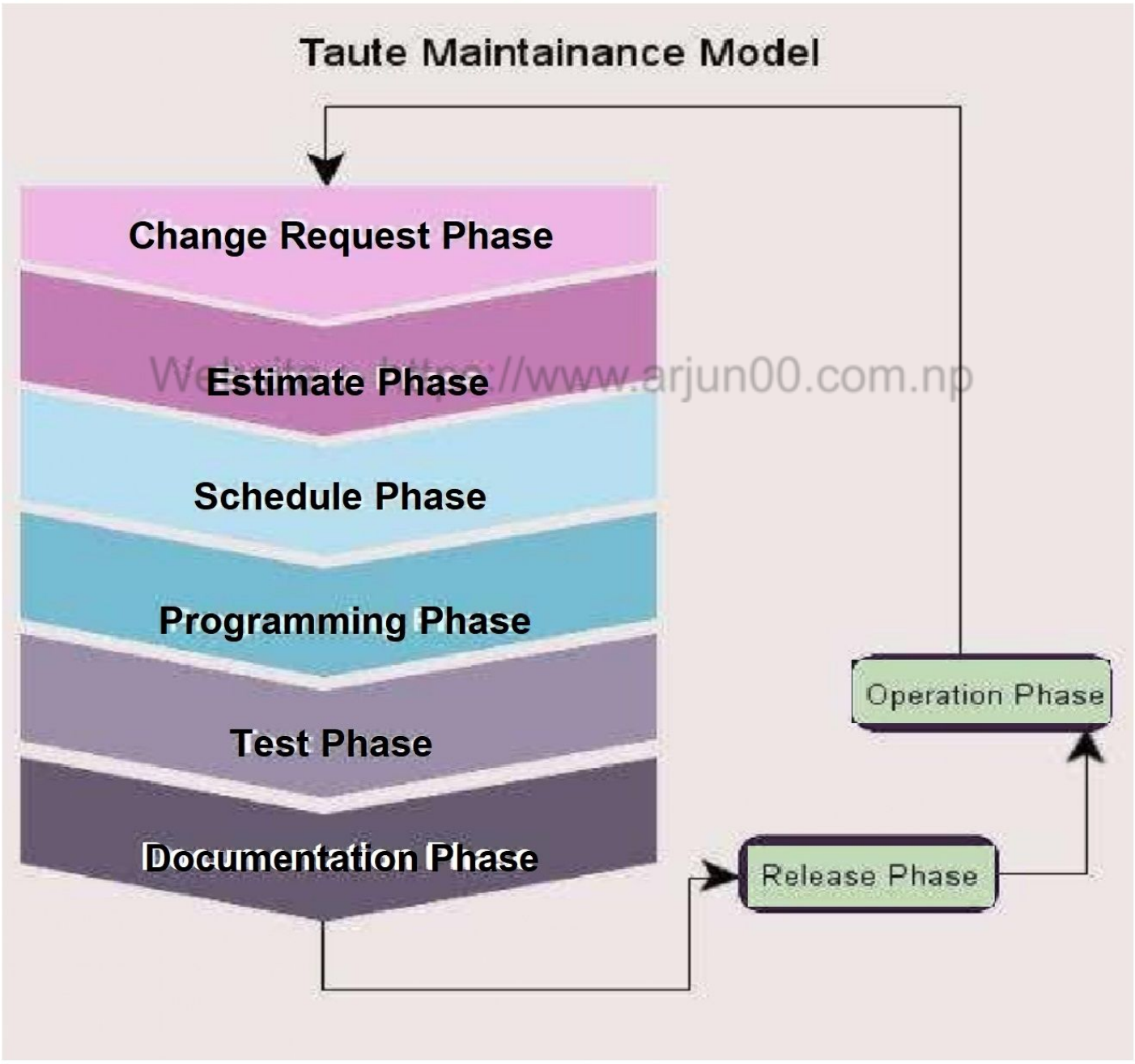
- Analysis of software system.
- Classification of requested modifications.
- Implementation of requested modifications.
- <u>The Re-use Oriented Model:</u>
- The parts of the old/existing system that are appropriate for reuse are identified and understood, in Reuse Oriented Model. These parts are then go through modification and enhancement, which are done on the basis of the specified new requirements. The final step of this model is the integration of modified parts into the new system.



- Boehm's Model:
- **Boehm's Model** performs maintenance process based on the economic models and principles. It represents the maintenance process in a closed loop cycle, wherein changes are suggested and approved first and then are executed.



- Taute Maintenance Model:
- Named after the person who proposed the model, Taute's model is a typical maintenance model that consists of eight phases in cycle fashion. The process of maintenance begins by requesting the change and ends with its operation. The phases of **Taute's Maintenance Model** are:



- **Change request Phase.**
- **Estimate Phase.**
- **Schedule Phase.**
- **Programming Phase.**
- **Test Phase.**
- **Documentation Phase.**
- **Release Phase.**
- **Operation Phase.**

Software maintenance cost

- **Software maintenance cost factors:**
  The key factors that distinguish development and maintenance and which lead to higher maintenance cost are divided into two subcategories:
- Non-Technical factors
- Technical factors

- **Non-Technical factors:**
  The Non-Technical factors include:
- Application Domain
- Staff stability
- Program lifetime
- Dependence on External Environment
- Hardware stability

- **Technical factors:**
  Technical factors include the following:
- module independence
- Programming language
- Programming style
- Program validation and testing
- Documentation
- Configuration management techniques

- *Efforts* expanded on maintenance may be divided into productivity activities (for example analysis and evaluation, design and modification, coding). The following expression provides a module of maintenance efforts:
- $M = P + K(C - D)$
- where,
  M: Total effort expanded on the maintenance.
  P: Productive effort.
  K: An empirical constant.
  C: A measure of complexity that can be attributed to a lack of good design and documentation.
  D: A measure of the degree of familiarity with the software.

Quality assurance

- 10.1 Software quality attributes
- 10.2 Quality factors
- 10.3 Quality control
- 10.4 Quality assurance
- 10.5 Software quality assurance
- 10.6 Software safety
- 10.7 The ISO 9000 model
- 10.8 SEI capability maturity model
- 10.9 Verification and validation

- Software Quality Attributes are features that facilitate the measurement of **performance of a software product** by Software Testing professionals, and include attributes such as availability, interoperability, correctness, reliability, learnability, robustness, maintainability, readability, extensibility, testability, efficiency, and portability. High scores in **Software Quality Attributes** enable software architects to guarantee that a software application will perform as the specifications provided by the client.
- **#1) Reliability**
- Measure if the product is reliable enough to sustain in any condition. Should give the correct results consistently. Product reliability is measured in terms of working of the project under different working environments and different conditions.

- **#2) Maintainability**
- Different versions of the product should be easy to maintain. For development, it should be easy to addcode to the existing system, should be easy to upgrade for new features and new technologies from time to time.

- **#3) Usability**
- This can be measured in terms of ease of use. The application should be user-friendly. It should be easy to learn. Navigation should be simple.
- **#4) Portability**
- This can be measured in terms of Costing issues related to porting, Technical issues related to porting, and Behavioral issues related to porting.

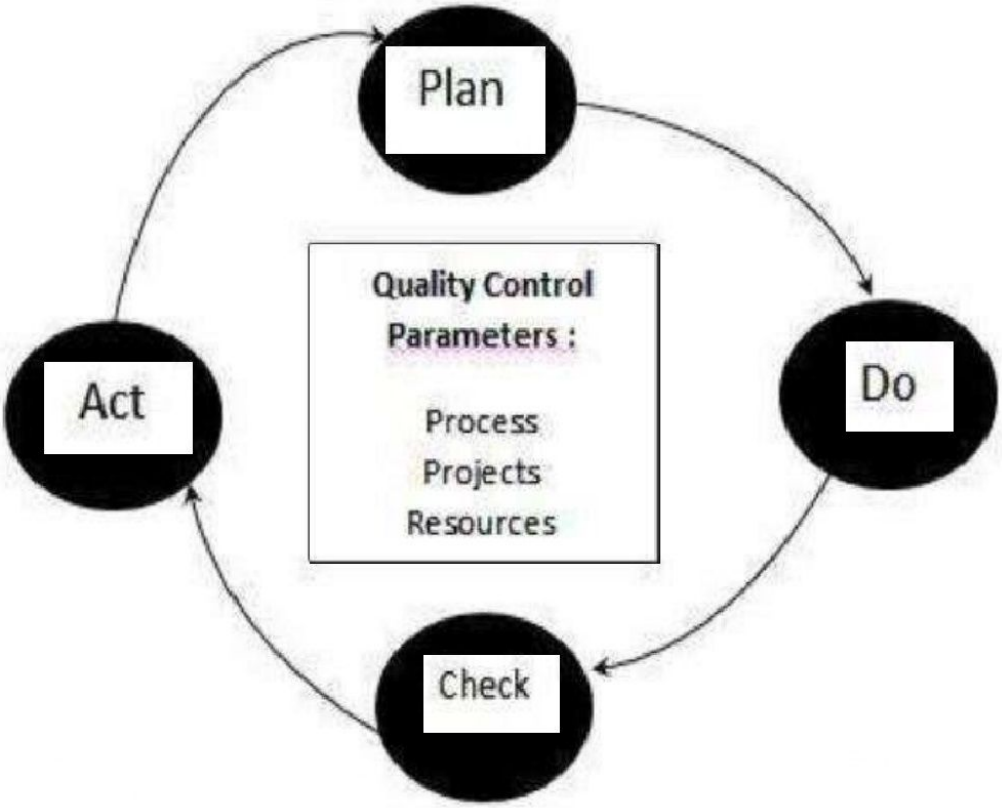
- **#5) Correctness**
- The application should be correct in terms of its functionality, calculations used internally and the navigation should be correct. This means that the application should adhere to functional requirements.
- **#6) Efficiency**
- It is one of the major system quality attributes. It is measured in terms of time required to complete any task given to the system. **For example,** the system should utilize processor capacity, disk space, and memory efficiently.

- **#7) Integrity or Security**
- Integrity comes with security. System integrity or security should be sufficient to prevent unauthorized access to system functions, prevent information loss, ensure that the software is protected from virus infection, and protect the privacy of data entered into the system.
- **#8) Testability**
- The system should be easy to test and find defects. If required, it should be easy to divide into different modules for testing.

- **#9) Interoperability**
- Interoperability of one system to another should be easy for the product to exchange data or services with other systems. Different system modules should work on different operating system platforms, different databases, and protocol conditions.

- What is Quality Control?
- Quality control is a set of methods used by organizations to achieve quality parameters or quality goals and continually improve the organization's ability to ensure that a software product will meet quality goals.

Quality Control Process:



- The three class parameters that control software quality are:
- Products
- Processes
- Resources
- The total quality control process consists of:
- Plan - It is the stage where the Quality control processes are planned
- Do - Use a defined parameter to develop the quality
- Check - Stage to verify if the quality of the parameters are met
- Act - Take corrective action if needed and repeat the work

Quality Control characteristics:

- Process adopted to deliver a quality product to the clients at best cost.
- Goal is to learn from other organizations so that quality would be better each time.
- To avoid making errors by proper planning and execution with correct review process.

- **Quality Assurance:** Quality Assurance is the preventive set of activities that provide greater confidence that the project will be completed successfully.
- **Quality Assurance** focuses on how the engineering and management activity will be done?
- As anyone is interested in the quality of the final product, it should be assured that we are building the right product.
- It can be assured only when we do inspection & review of intermediate products, if there are any bugs, then it is debugged. This quality can be

Software Quality Assurance

- Software quality assurance is a planned and systematic plan of all actions necessary to provide adequate confidence that an item or product conforms to establish technical requirements.
- A set of activities designed to calculate the process by which the products are developed or manufactured.

SQA Encompasses

- A quality management approach
- Effective Software engineering technology (methods and tools)
- Formal technical reviews that are tested throughout the software process
- A multitier testing strategy
- Control of software documentation and the changes made to it.
- A procedure to ensure compliances with software development standards
- Measuring and reporting mechanisms.

- SQA Activities
- Software quality assurance is composed of a variety of functions associated with two different constituencies ? the software engineers who do technical work and an SQA group that has responsibility for quality assurance planning, record keeping, analysis, and reporting.

- **Following activities are performed by an independent SQA group:**
- **Prepares an SQA plan for a project:** The program is developed during project planning and is reviewed by all stakeholders.
- **Participates in the development of the project's software process description:** The software team selects a process for the work to be performed.
- **Reviews software engineering activities to verify compliance with the defined software process:** The SQA group identifies, reports, and tracks deviations from the process and verifies that corrections have been made.
- **Audits designated software work products to verify compliance with those defined as a part of the software process:**

- **Ensures that deviations in software work and work products are documented and handled according to a documented procedure:** Deviations may be encountered in the project method, process description, applicable standards, or technical work products.
- **Records any noncompliance and reports to senior management:** Non- compliance items are tracked until they are resolved.

Quality Assurance v/s Quality control

Quality Assurance	Quality Control
Quality Assurance (QA) is the set of actions including facilitation, training, measurement, and analysis needed to provide adequate confidence that processes are established and continuously improved to produce products or services that conform to specifications and are fit for use.	**Quality Control (QC)** is described as the processes and methods used to compare product quality to requirements and applicable standards, and the actions are taken when a nonconformance is detected.
QA is an activity that establishes and calculates the processes that produce the product. If there is no process, there is no role for QA.	**QC** is an activity that demonstrates whether or not the product produced met standards.
QA helps establish process	**QC** relates to a particular product or service
QA sets up a measurement program to evaluate processes	**QC** verified whether particular attributes exist, or do not exist, in a explicit product

QA sets up a measurement program to evaluate processes	**QC** verified whether particular attributes exist, or do not exist, in a explicit product or service.
QA identifies weakness in processes and improves them	**QC** identifies defects for the primary goals of correcting errors.
Quality Assurance is a managerial tool.	Quality Control is a corrective tool.
Verification is an example of QA.	Validation is an example of QC.

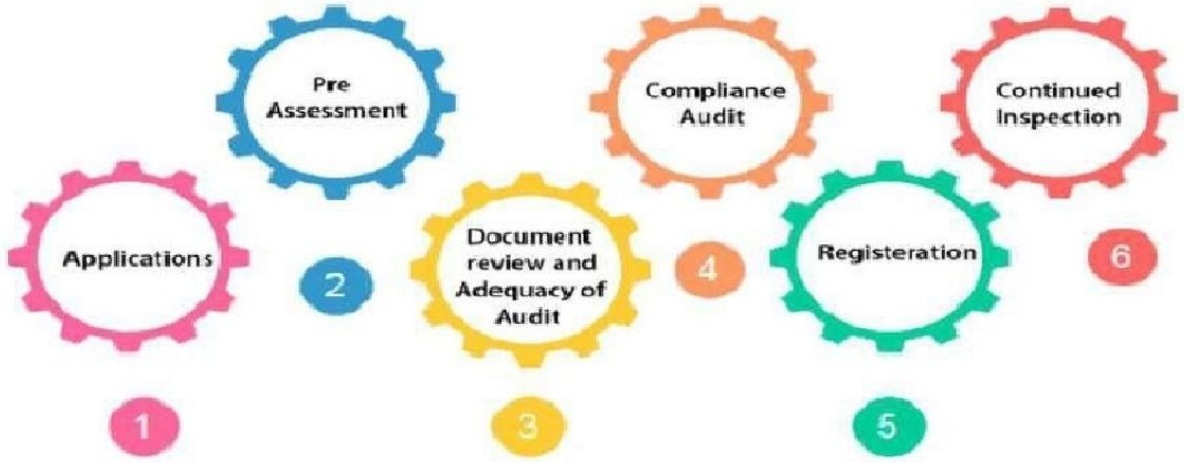
The ISO 9000 model

- ISO (International Standards Organization) is a group or consortium of 63 countries established to plan and fosters standardization. ISO declared its 9000 series of standards in 1987. It serves as a reference for the contract between independent parties. The ISO 9000 standard determines the guidelines for maintaining a quality system. The ISO standard mainly addresses operational methods and organizational methods such as responsibilities, reporting, etc. ISO 9000 defines a set of guidelines for the production process and is not directly concerned about the product itself.

Types of ISO 9000 Quality Standards

ISO 9000 is a series of three standards:

ISO 9000 Certification



- **Application:** Once an organization decided to go for ISO certification, it applies to the registrar for registration.
- **Pre-Assessment:** During this stage, the registrar makes a rough assessment of the organization.
- **Document review and Adequacy of Audit:** During this stage, the registrar reviews the document submitted by the organization and suggest an improvement.
- **Compliance Audit:** During this stage, the registrar checks whether the organization has compiled the suggestion made by it during the review or not.
- **Registration:** The Registrar awards the ISO certification after the successful completion of all the phases.
- **Continued Inspection:** The registrar continued to monitor the organization time by time.

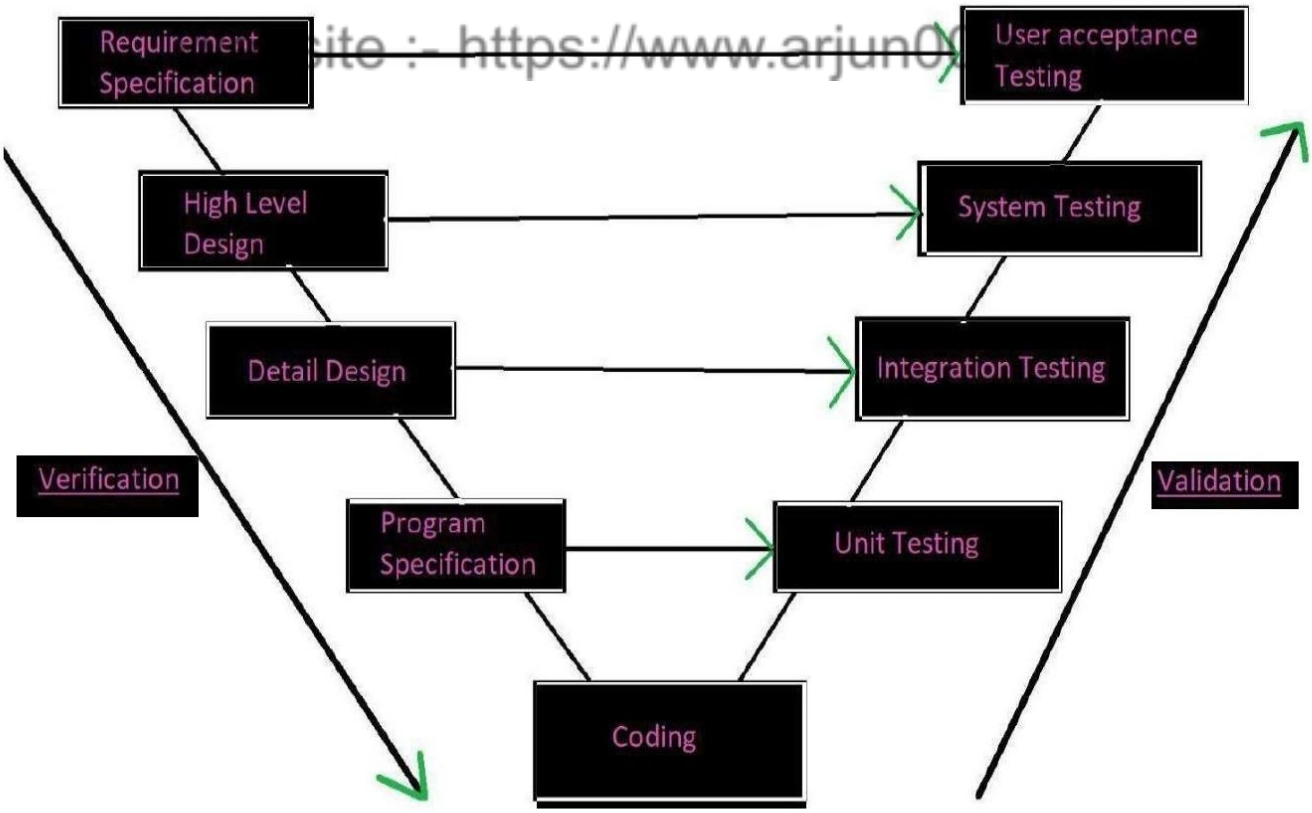
- Verification and Validation is the process of investigating that a software system satisfies specifications and standards and it fulfills the required purpose. **Barry Boehm** described verification and validation as the following:
- **Verification:** Are we building the product right?
  Validation: Are we building the right product?

- The ISO 9000 series of standards is based on the assumption that if a proper stage is followed for production, then good quality products are bound to follow automatically. The types of industries to which the various ISO standards apply are as follows.
- **ISO 9001:** This standard applies to the organizations engaged in design, development, production, and servicing of goods. This is the standard that applies to most software development organizations.
- **ISO 9002:** This standard applies to those organizations which do not design products but are only involved in the production. Examples of these category industries contain steel and car manufacturing industries that buy the product and plants designs from external sources and are engaged in only manufacturing those products. Therefore, ISO 9002 does not apply to software development organizations.

- **ISO 9003:** This standard applies to organizations that are involved only in the installation and testing of the products. For example, Gas companies.
- How to get ISO 9000 Certification?
- An organization determines to obtain ISO 9000 certification applies to ISO registrar office for registration. The process consists of the following stages:

- **Verification:**
  Verification is the process of checking that a software achieves its goal without any bugs. It is the process to ensure whether the product that is developed is right or not. It verifies whether the developed product fulfills the requirements that we have.
  Verification is **Static Testing**.
- Activities involved in verification:
- Inspections
- Reviews
- Walkthroughs
- Desk-checking

- **Validation:**
  Validation is the process of checking whether the software product is up to the mark or in other words product has high level requirements. It is the process of checking the validation of product i.e. it checks what we are developing is the right product. it is validation of actual and expected product.
  Validation is the **Dynamic Testing**.
- Activities involved in validation:
- Black box testing
- White box testing
- Unit testing
- Integration testing



Software Safety

- **Software Safety**
- Software has been built into more and more products and systems over the years and has taken on more and more of the functionality of those systems. The question is: how dependable is the functionality provided by software? The traditional approach to verification of functionality - try it out and see if it works - is of limited value in the case of software which can be much more complex than hardware.
- What is Safety Testing?
- Safety testing in software systems aims at optimizing system safety in the design, development, use, and maintenance of software systems and their integration with safety-critical hardware systems in a production environment.

- Aspects of Software Safety:
- Functioning software should not generate hazards - Eg: Guiding the state of the art aircraft should NOT steer into the ocean
- Monitoring systems must perform flawlessly - Eg: Back-up computer Should start automatically when primary fails
- Goals in Safety Testing:
- In complex systems where there are many interactions involved, the safety-critical functionality should be identified and thoroughly analyzed.
- Contributing factors and resultant hazards associated with the system are identified and eliminated.
- The number of safety critical interfaces are kept low to avoid injury or death.
- Safety attributes are to be addressed as part of all the levels of software testing.