# Chapter 1
# Introduction to Software Engineering

## Definition of software:

Software is a general term used to describe a collection of computer programs, Procedures and documentation that performs real business problems. In simple word you can say software is the way to perform different tasks electronically.

Software is computer programs and associated documentation. This definition clearly states that, the software is not a collection of programs, but includes all associated documentation.

Software system usually consists of a number of separate programs, configuration files: which are used to set up these programs, System documentation: which describes the structure of the system, and user documentation: which explains how to use the system and web sites for users to download recent product information.

Software products may be developed for a particular customer or may be developed for a general market

## The Evolving Role of Software:

- Software takes on a dual role: product and at the same time, the vehicle for delivering a product.
- As a product, it delivers the computing potential embodied by computer hardware, or more broadly, a network of computers that are accessible by local hardware.
- As the vehicle used to deliver the product, software acts as the basis for the control of the computer, the communication of information, and the creation and control of other programs.
- The role of computer software has undergone significant change over a time span of little more than 50 years.
- Dramatic improvements in hardware performance, profound changes in computing architectures, vast increases in memory and storage capacity, and a wide variety of exotic input and output options have all precipitated more sophisticated and complex computer based systems.
- The programmers are asked some questions when modern computer-based systems are built:
    - Why does it take so long to get software finished?
    - Why are development costs so high?
    - Why can't we find all the errors before we give the software to customer?
    - Why do we continue to have difficulty in measuring progress as software is being developed?





## Changing Nature of Software:

Software can be applied in any situation for which pre-specified set of procedural steps has been defined. Software can be applied on countless field. For e.g.: Business, education, social sector etc. Software is designed to suit some specific goal. Software is classified according to the range of potential of applications.

[➔ was discussed during the class and types were given to the end of this chapter]

## Characteristics of Software:

1. **Software is developed or engineered, it is not manufactured.**
    - Although some similarities exist between software development and hardware manufacture, the two activities are fundamentally different.
    - In both activities, high quality is achieved through good design, but the manufacturing phase for hardware can introduce quality problems that are nonexistent (or easily corrected) for software.
    - Software costs are concentrated in engineering.
    - Software projects cannot be managed as if they were manufacturing projects.

2. **Software development presents a job shop environment.**
3. **Software does not wear out.**
    - The failure rate of the hardware is high as the hardware components suffer from the
    - Cumulative effects of dust, vibration, abuse, temperature extremes, and many other environmental maladies.
    - Stated simply, the hardware begins to wear out.
    - Software is not susceptible to the environmental maladies that cause hardware to wear out.
    - Undiscovered defects will cause high failure rates early in the life of a program.
    - However, these are corrected ideally, without introducing other errors
    - Software does not wear out, but it deteriorates.
    - Software maintenance involves considerably more complexity than hardware maintenance.
4. **Time and effort for software development are hard to estimate.**
5. **Testing of software is extremely difficult.**
6. **Software development can't address each and every requirements of customer.**
7. **It is very difficult to manage software and hardware simultaneously.**

## A Generic view of Software engineering:

### 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.

### 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.

### 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, that is, CPU, operating system, business rules etc 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.





## Software Engineering layered technology:


Fig: Software Engineering as a layered technology

**a) A quality Process :-**
- Any engineering approach must rest on an quality.
- The "Bed Rock" that supports software Engineering is Quality Focus.

**b) Process :-**
- Foundation for SE is the Process Layer
- SE process is the GLUE that holds all the technology layers together and enables the timely development of computer software.
- It forms the base for management control of software project.

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

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

## Classification (Types) of Software:

Software can be applied in countless fields such as business, education, social sector, and other fields. It is designed to suit some specific goals such as data processing, information sharing, communication, and so on. It is classified according to the range of potential of applications. These classifications are listed below.

### a. System software:

- System software is a collection of programs written to service other programs.
- Some system software (e.g. compilers, editors, and file management utilities) process complex, but determinate, information structures.
- Other system applications (e.g. drivers, operating system components etc.) process largely intermediate data.
- In either case, the system software area is characterized by heavy interaction with computer
- For example, an operating system is a system software, which controls the hardware, manages memory and multitasking functions, and acts as an interface between application programs and the computer.

### b. Real-time software:

- Software that monitors, analyzes, controls real world events as they occur is called real time software.
- Elements of real time software include a data gathering component that collects and formats information from an external environment, a control output component that coordinates all other components so that real-time response can be maintained.

  An example of real-time software is the software used for weather forecasting that collects and processes parameters like temperature and humidity from the external environment to forecast the weather. Most of the defense organizations all over the world use real-time software to control their military hardware.



### c. Business software:

- Business information processing is the largest single software application area.
- In addition to conventional data processing application business software applications also encompass interactive computing.
- Applications in this area restructure existing data in a way that facilitates business operations or management decision making.





### d. Engineering and scientific software:

Engineering and scientific software have been characterized by "number crunching" algorithms. Applications range from astronomy to volcano-logy and from molecular biology to automated manufacturing.

### e. Artificial intelligence (AI) software:

This class of software is used where the problem-solving technique is non-algorithmic in nature. The solutions of such problems are generally non-agreeable to computation or straightforward analysis. Instead, these problems require specific problem-solving strategies that include expert system, pattern recognition, and game-playing techniques. In addition, they involve different kinds of search techniques which include the use of heuristics. The role of artificial intelligence software is to add certain degrees of intelligence to the mechanical hardware in order to get the desired work done in an agile manner.

### f. Web-based software:

This class of software acts as an interface between the user and the Internet. Data on the Internet is in the form of text, audio, or video format, linked with hyperlinks. Web browser is a software that retrieves web pages from the Internet. The software incorporates executable instructions written in special scripting languages such as CGI or ASP. Apart from providing navigation on the Web, this software also supports additional features that are useful while surfing the Internet.

### g. Personal computer (PC) software:

This class of software is used for both official and personal use. The personal computer software market has grown over in the last two decades from normal text editor to word processor and from simple paintbrush to advanced image-editing software. This software is used predominantly in almost every field, whether it is database management system, financial accounting package, or multimedia-based software. It has emerged as a versatile tool for routine applications.

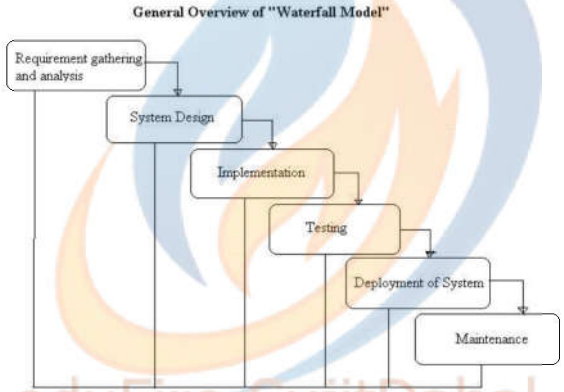
End of Chapter 1

# Chapter: 2
# Process Model

## Waterfall Model:

The Waterfall Model was first Process Model to be introduced. It is also referred to as a **linear-sequential life cycle model.** It is very simple to understand and use. In a waterfall model, each phase must be completed fully before the next phase can begin. This type of model is basically used for the for the project which is small and there are no uncertain requirements. At the end of each phase, a review takes place to determine if the project is on the right path and whether or not to continue or discard the project. In this model the testing starts only after the development is complete. In **waterfall model phases** do not overlap.

### Diagram of Waterfall-model:






### Advantages of waterfall model:

- This model is simple and easy to understand and use.
- It is easy to manage due to the rigidity of the model – each phase has specific deliverables and a review process.
- In this model phases are processed and completed one at a time. Phases do not overlap.
- Waterfall model works well for smaller projects where requirements are very well understood.





### Disadvantages of waterfall model:

- Once an application is in the testing stage, it is very difficult to go back and change something that was not well-thought out in the concept stage.
- No working software is produced until late during the life cycle.
- High amounts of risk and uncertainty.
- Not a good model for complex and object-oriented projects.
- Poor model for long and ongoing projects.
- Not suitable for the projects where requirements are at a moderate to high risk of changing.

### When to use the waterfall model:

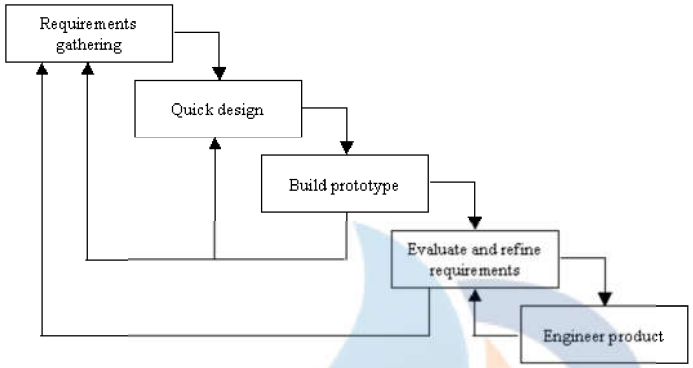
- This model is used only when the requirements are very well known, clear and fixed.
- Product definition is stable.
- Technology is understood.
- There are no ambiguous requirements
- Ample resources with required expertise are available freely
- The project is short.

Very less customer enter action is involved during the development of the product. Once the product is ready then only it can be demoed to the end users. Once the product is developed and if any failure occurs then the cost of fixing such issues are very high, because we need to update everywhere from document till the logic.

## Prototyping Model:

The basic idea here is that instead of freezing the requirements before a design or coding can proceed, a throwaway prototype is built to understand the requirements. This prototype is developed based on the currently known requirements. By using this prototype, the client can get an "actual feel" of the system, since the interactions with prototype can enable the client to better understand the requirements of the desired system. Prototyping is an attractive idea for complicated and large systems for which there is no manual process or existing system to help determining the requirements. The prototype are usually not complete systems and many of the details are not built in the prototype. The goal is to provide a system with overall functionality.

### Diagram of Prototype model:




### Advantages of Prototype model:

- Users are actively involved in the development
- Since in this methodology a working model of the system is provided, the users get a better understanding of the system being developed.
- Errors can be detected much earlier.
- Quicker user feedback is available leading to better solutions.
- Missing functionality can be identified easily
- Confusing or difficult functions can be identified Requirements validation, Quick implementation of, incomplete, but functional, application.

### Disadvantages of Prototype model:

- Leads to implementing and then repairing way of building systems.
- Practically, this methodology may increase the complexity of the system as scope of the system may expand beyond original plans.
- Incomplete application may cause application not to be used as the full system was designed Incomplete or inadequate problem analysis.

### When to use Prototype model:

- Prototype model should be used when the desired system needs to have a lot of interaction with the end users.
- Typically, online systems, web interfaces have a very high amount of interaction with end users, are best suited for Prototype model. It might take a while for a system to be built that allows ease of use and needs minimal training for the end user.





- Prototyping ensures that the end users constantly work with the system and provide a feedback which is incorporated in the prototype to result in a useable system. They are excellent for designing good human computer interface systems.

## RAD(Rapid Application Development) Model:

RAD model is Rapid Application Development model. It is a type of incremental model. In RAD model the components or functions are developed in parallel as if they were mini projects. The developments are time boxed, delivered and then assembled into a working prototype. This can quickly give the customer something to see and use and to provide feedback regarding the delivery and their requirements.

### Diagram of RAD-Model:

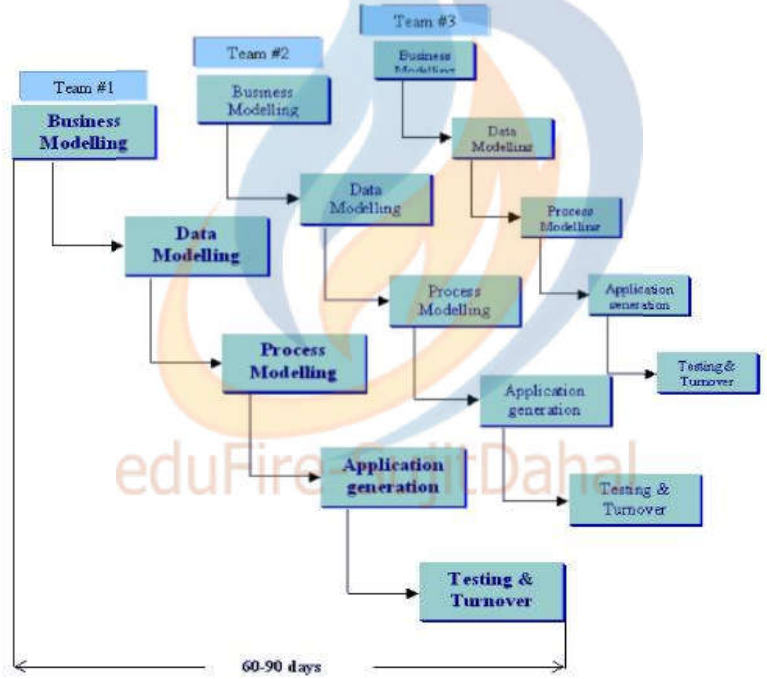



Figure 1.5 – RAD Model

The phases in the rapid application development (RAD) model are:

1. **Business modeling:** The information flow is identified between various business functions.
2. **Data modeling:** Information gathered from business modeling is used to define data objects that are needed for the business.
3. **Process modeling:** Data objects defined in data modeling are converted to achieve the business information flow to achieve some specific business objective. Description are identified and created for CRUD of data objects.
4. **Application generation:** Automated tools are used to convert process models into code and the actual system.
5. **Testing and turnover:** Test new components and all the interfaces.

### Advantages of the RAD model:

- Reduced development time.
- Increases reusability of components
- Quick initial reviews occur
- Encourages customer feedback
- Integration from very beginning solves a lot of integration issues.

### Disadvantages of RAD model:

- Depends on strong team and individual performances for identifying business requirements.
- Only system that can be modularized can be built using RAD
- Requires highly skilled developers/designers.
- High dependency on modeling skills
- Inapplicable to cheaper projects as cost of modeling and automated code generation is very high.

### When to use RAD model:

- RAD should be used when there is a need to create a system that can be modularized in 2-3 months of time.
- It should be used if there's high availability of designers for modeling and the budget is high enough to afford their cost along with the cost of automated code generating tools.
- RAD SDLC model should be chosen only if resources with high business knowledge are available and there is a need to produce the system in a short span of time (2-3 months).







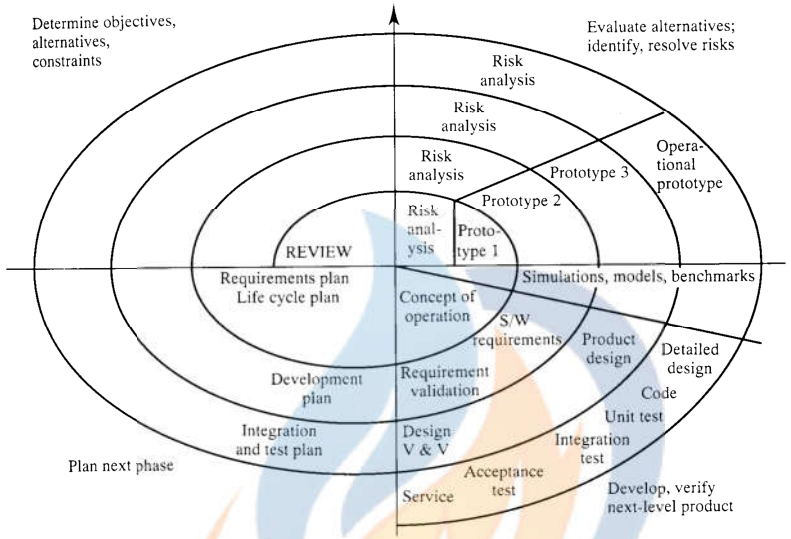
## Spiral Model:

The spiral model is similar to the incremental model, with more emphasis placed on risk analysis. The spiral model has four phases: Planning, Risk Analysis, Engineering and Evaluation. A software project repeatedly passes through these phases in iterations (called Spirals in this model). The baseline spiral, starting in the planning phase, requirements are gathered and risk is assessed. Each subsequent spirals builds on the baseline spiral.

Spiral model is a combination of both, iterative model and one of the SDLC model. It can be seen as if you choose one SDLC model and combine it with cyclic process (iterative model).

a) **Planning Phase:** Requirements are gathered during the planning phase. Requirements like 'BRS' that is 'Business Requirement Specifications' and 'SRS' that is 'System Requirement specifications'.
b) **Risk Analysis:** In the risk analysis phase, a process is undertaken to identify risk and alternate solutions. A prototype is produced at the end of the risk analysis phase. If any risk is found during the risk analysis then alternate solutions are suggested and implemented.
c) **Engineering Phase:** In this phase software is developed, along with testing at the end of the phase. Hence in this phase the development and testing is done.
d) **Evaluation phase:** This phase allows the customer to evaluate the output of the project to date before the project continues to the next spiral.

## Diagram of Spiral model:




## Advantages of Spiral model:

- High amount of risk analysis hence, avoidance of Risk is enhanced.
- Good for large and mission-critical projects.
- Strong approval and documentation control.
- Additional Functionality can be added at a later date.
- Software is produced early in the **software life cycle**.



## Disadvantages of Spiral model:

- Can be a costly model to use.
- Risk analysis requires highly specific expertise.
- Project's success is highly dependent on the risk analysis phase.
- Doesn't work well for smaller projects.

## When to use Spiral model:

- When costs and risk evaluation is important
- For medium to high-risk projects
- Long-term project commitment unwise because of potential changes to economic priorities
- Users are unsure of their needs





- Requirements are complex
- New product line
- Significant changes are expected (research and exploration)

*** End of Chapter 2 ***

# Chapter: 3
# Software Project Management

## Software Project Management(SPM)

Software project management is the art and science of planning and leading software projects. It is a sub-discipline of project management in which software projects are planned, implemented, monitored and controlled.

## Meaning of 4P's in SPM

### The People:

The primary element of any project is the people. People gather requirements, people interview users (people), people design software, and people write software for people. No people -- no software.

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

### The Product:

The product is the result of a project. The desired product satisfies the customers and keeps them coming back for more. Sometimes, however, the actual product is something less.

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

### The Process:

Process is how we go from the beginning to the end of a project. All projects use a process. Many project managers, however, do not choose a process based on the people and product at hand. They simply use the same process they've always used or misused.

A software process provides the framework from which a comprehensive plan for software development can be established. A number of different tasks sets— tasks, milestones, work





products, and quality assurance points—enable the framework activities to be adapted to the characteristics of the software project and the requirements of the project team. Finally, umbrella activities overlay the process model. Umbrella activities are independent of any one framework activity and occur throughout the process.

### The Project

The product is the result of a project. The desired product satisfies the customers and keeps them coming back for more. Sometimes, however, the actual product is something less. The product pays the bills and ultimately allows people to work together in a process and build software

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

## Activities of Project Planning

[➔ was discussed during the class]

## Project Estimation Technique

[➔ was discussed during the class]

## Project Scheduling Techniques:

1) Work Breakdown Structure (WBS)
2) Activity Chart
3) Gantt Charts
4) PERT
5) Critical Path Method (CPM)

### 1. Work Breakdown Structure (WBS):

A work breakdown structure (WBS), in project management and systems engineering, is a deliverable-oriented decomposition of a project into smaller components. A work breakdown structure is a key project deliverable that organizes the team's work into manageable sections.

Dividing complex projects to simpler and manageable tasks is the process identified as Work Breakdown Structure (WBS).

Usually, the project managers use this method for simplifying the project execution. In WBS, much larger tasks are broken down to manageable chunks of work. These chunks can be easily supervised and estimated. WBS is not restricted to a specific field when it comes to application. This methodology can be used for any type of project management.

Following are a few reasons for creating a WBS in a project:

- Accurate and readable project organization.
- Accurate assignment of responsibilities to the project team.
- Indicates the project milestones and control points.
- Helps to estimate the cost, time and risk.
- Illustrate the project scope, so the stakeholders can have a better understanding of the same.

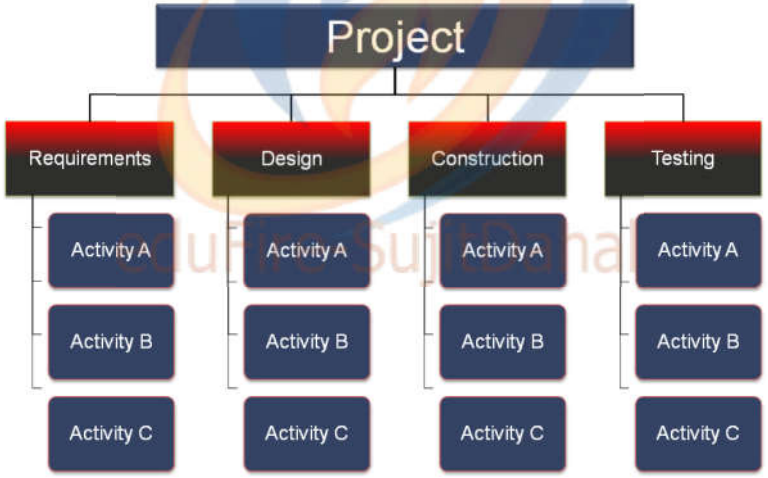
Construction of a WBS

Identifying the main deliverables of a project is the starting point for deriving a work breakdown structure.

This important step is usually done by the project managers and the subject matter experts (SMEs) involved in the project. Once this step is completed, the subject matter experts start breaking down the high-level tasks into smaller chunks of work.

In the process of breaking down the tasks, one can break them down into different levels of detail. One can detail a high-level task into ten sub-tasks while another can detail the same high-level task into 20 sub-tasks.

Therefore, there is no hard and fast rule on how you should breakdown a task in WBS. Rather, the level of breakdown is a matter of the project type and the management style followed for the project. In general, there are a few "rules" used for determining the smallest task chunk. In "two




weeks" rule, nothing is broken down smaller than two week worth of work.





This means, the smallest task of the WBS is at least two-week long. 8/80 is another rule used when creating a WBS. This rule implies that no task should be smaller than 8 hours of work and should not be larger than 80 hours of work.

One can use many forms to display their WBS. Some use tree structure to illustrate the WBS, while others use lists and tables. Outlining is one of the easiest ways of representing a WBS.

Conclusion

The efficiency of a work breakdown structure can determine the success of a project. The WBS provides the foundation for all project management work, including, planning, cost and effort estimation, resource allocation, and scheduling. Therefore, one should take creating WBS as a critical step in the process of project management.

## 2. Activity Chart:

When it comes to a project, the entire project is divided into many interdependent tasks. In this set of tasks, the sequence or the order of the tasks is quite important.

If the sequence is wrong, the end result of the project might not be what the management expected.

Some tasks in the projects can safely be performed parallel to other tasks. In a project activity diagram, the sequence of the tasks is simply illustrated.

There are many tools that can be used for drawing project activity diagrams. Microsoft Project is one of the most popular software for this type of work.

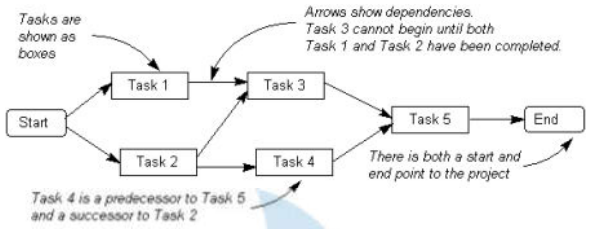
In addition to that, Microsoft Vision (for Windows) and Omni Graffle (for Mac) can be used to draw activity diagrams.

The Workflow

Have you seen process flow diagrams? If yes, then activity diagrams takes the same shape. Usually there are two main shapes in activity diagrams, boxes and arrows.

Boxes of the activity diagram indicate the tasks and the arrows show the relationships. Usually, the relationships are the sequences that take place in the activities.

Following is an example of activity diagram with tasks in boxes and relationship represented by arrows.




This type of activity diagram is also known as activity-on-node diagram. This is due to the fact that all activities (tasks) are shown on the nodes (boxes).

Alternatively, there is another way of presenting an activity diagram. This is called activity-on-arrow diagram. In this diagram, activities (tasks) are presented by the arrows.

Compared to activity-on-node diagrams, activity-on-arrow diagrams introduce a little confusion. Therefore, in most instances, people often use activity-on-nodes diagrams. Following is an activity-on-arrow diagram:

How to Draw Activity Diagram?

Creating an activity diagram is easy. You can use a paper-based material such as a post it note or software for this purpose. Regardless of the medium used, the process of creating the activity diagram remains the same.

Following are main steps involved in creating an activity diagram:

Step 1: First of all, identify the tasks in the project. You can use WBS (Work Breakdown Structure) for this purpose and there is no need to repeat the same.

Just use the same tasks breakdown for the activity diagram as well. If you use software for creating the activity diagram (which is recommended), create a box for each activity.

Illustrate all boxes in the same size in order to avoid any confusion. Make sure all your tasks have the same granularity.

Step 2: You can add more information to the task boxes, such as who is doing the task and the timeframes. You can add this information inside the box or can add it somewhere near the box.

Step 3: Now, arrange the boxes in the sequence that they are performed during the project execution. The early tasks will be at the left hand side and the tasks performed at the later part of





the project execution will be at the right hand side. The tasks that can be performed in parallel should be kept parallel to each other (vertically).

You may have to adjust the sequence a number of times until you get it right. This is why software is an easy tool for creating activity diagrams.

Step 4: Now, use arrows to join task boxes. These arrows will show the sequence of the tasks. Sometimes, a 'start' and an 'end' box can be added to clearly present the start and the end of the project.

Conclusion

Activity diagrams can be used for illustrating the sequence of project tasks. These diagrams can be created with a minimum effort and gives you a clear understanding of interdependent tasks.

In addition, the activity diagram is an input for the critical path method.

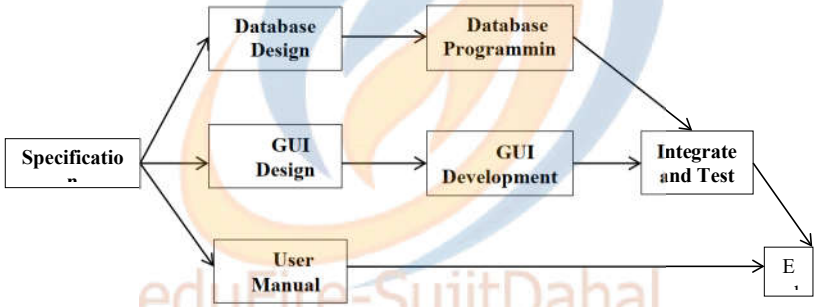



Fig: Activity Network (Chart) representation of MIS

## 3. Gantt Chart:

Gantt chart is a type of a bar chart that is used for illustrating project schedules. Gantt charts can be used in any projects that involve effort, resources, milestones and deliveries.

At present, Gantt charts have become the popular choice of project managers in every field.

Gantt charts allow project managers to track the progress of the entire project. Through Gantt charts, the project manager can keep a track of the individual tasks as well as of the overall project progression.

In addition to tracking the progression of the tasks, Gantt charts can also be used for tracking the utilization of the resources in the project. These resources can be human resources as well as materials used.

Gantt chart was invented by a mechanical engineer named Henry Gantt in 1910. Since the invention, Gantt chart has come a long way. By today, it takes different forms from simple paper based charts to sophisticated software packages.

Advantages & Disadvantages

- The ability to grasp the overall status of a project and its tasks at once is the key advantage in using a Gantt chart tool. Therefore, upper management or the sponsors of the project can make informed decisions just by looking at the Gantt chart tool.
- The software-based Gantt charts are able to show the task dependencies in a project schedule. This helps to identify and maintain the critical path of a project schedule.
- Gantt chart tools can be used as the single entity for managing small projects. For small projects, no other documentation may be required; but for large projects, the Gantt chart tool should be supported by other means of documentation.
- For large projects, the information displayed in Gantt charts may not be sufficient for decision making.
- Although Gantt charts accurately represent the cost, time and scope aspects of a project, it does not elaborate on the project size or size of the work elements. Therefore, the magnitude of constraints and issues can be easily misunderstood.

Conclusion

Gantt chart tools make project manager's life easy. Therefore, Gantt chart tools are important for successful project execution.

Identifying the level of detail required in the project schedule is the key when selecting a suitable Gantt chart tool for the project.







One should not overly complicate the project schedules by using Gantt charts to manage the simplest tasks.

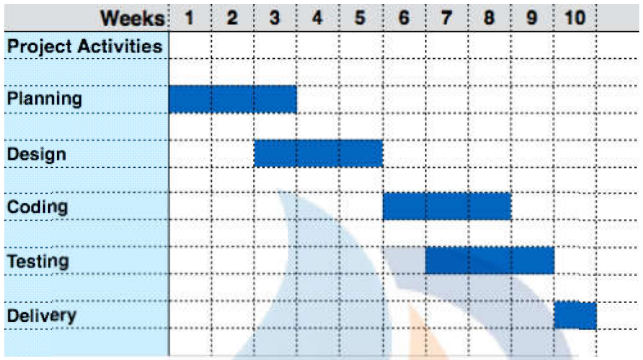



Fig: Gantt chart

## 4. PERT:

Before any activity begins related to the work of a project, every project requires an advanced, accurate time estimate. Without an accurate estimate, no project can be completed within the budget and the target completion date.

Developing an estimate is a complex task. If the project is large and has many stakeholders, things can be more complex.

Therefore, there have been many initiatives to come up with different techniques for estimation phase of the project in order to make the estimation more accurate.

PERT (Program Evaluation and Review Technique) is one of the successful and proven methods among the many other techniques, such as, CPM, Function Point Counting, Top-Down Estimating, WAVE, etc.

PERT was initially created by the US Navy in the late 1950s. The pilot project was for developing Ballistic Missiles and there have been thousands of contractors involved.

After PERT methodology was employed for this project, it actually ended two years ahead of its initial schedule.

The PERT Basics

At the core, PERT is all about management probabilities. Therefore, PERT involves in many simple statistical methods as well.

Sometimes, people categorize and put PERT and CPM together. Although CPM (Critical Path Method) shares some characteristics with PERT, PERT has a different focus.

Same as most of other estimation techniques, PERT also breaks down the tasks into detailed activities.

Then, a Gantt chart will be prepared illustrating the interdependencies among the activities. Then, a network of activities and their inter-dependencies are drawn in an illustrative manner.

In this map, a node represents each event. The activities are represented as arrows and they are drawn from one event to another, based on the sequence.

Next, the Earliest Time (TE) and the Latest Time (TL) are figured for each activity and identify the slack time for each activity.

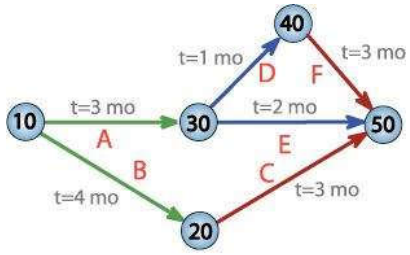
When it comes to deriving the estimates, the PERT model takes a statistical route to do that. We will cover more on this in the next two sections.

Following is an example PERT chart:










PERT network chart for a seven-month project with five milestones (10 through 50) and six activities (A through F)

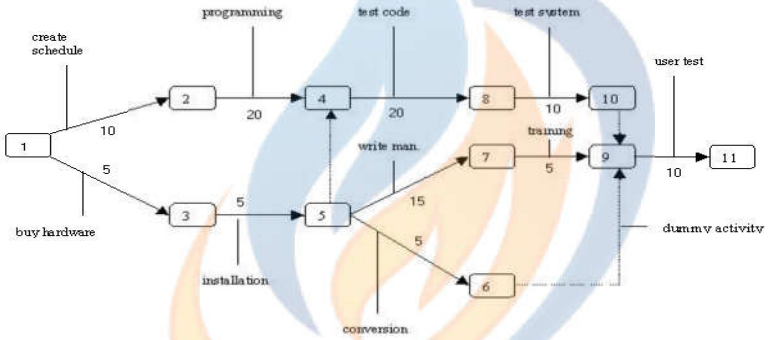



Fig. 1: PERT Chart

* Numbered rectangles are nodes and represent events or milestones.
* Directional arrows represent dependent tasks that must be completed sequentially.
* Diverging arrow directions (e.g. 1-2 & 1-3) indicate possibly concurrent tasks
* Dotted lines indicate dependent tasks that do not require resources.

## 5. Critical Path Method (CPM):

If you have been into project management, I'm sure you have already heard the term 'critical path method.'

If you are new to the subject, it is best to start with understanding the 'critical path' and then move on to the 'critical path method.'

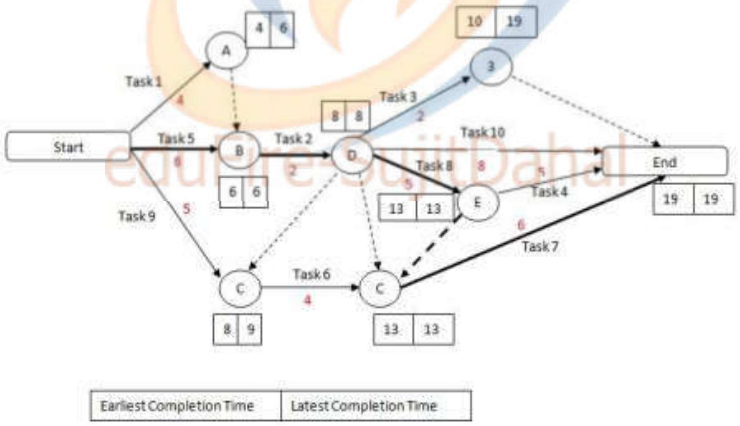
Critical path is the sequential activities from start to the end of a project. Although many projects have only one critical path, some projects may have more than one critical paths depending on the flow logic used in the project.

If there is a delay in any of the activities under the critical path, there will be a delay of the project deliverables.

Most of the times, if such delay is occurred, project acceleration or re-sequencing is done in order to achieve the deadlines.

Critical path method is based on mathematical calculations and it is used for scheduling project activities. This method was first introduced in 1950s as a joint venture between Remington Rand Corporation and DuPont Corporation.

The initial critical path method was used for managing plant maintenance projects. Although the original method was developed for construction work, this method can be used for any project where there are interdependent activities.








In the critical path method, the critical activities of a program or a project are identified. These are the activities that have a direct impact on the completion date of the project.

Advantages of Critical Path Method

Following are advantages of critical path methods:

- Offers a visual representation of the project activities.
- Presents the time to complete the tasks and the overall project.
- Tracking of critical activities.

Conclusion

Critical path identification is required for any project-planning phase. This gives the project management the correct completion date of the overall project and the flexibility to float activities.

A critical path diagram should be constantly updated with actual information when the project progresses in order to refine the activity length/project duration predictions.

## COCOMO-Model (Constructive Cost Model):

The Constructive Cost Model (COCOMO) is an algorithmic software cost estimation model developed by Barry Boehm. The model uses a basic regression formula, with parameters that are derived from historical project data and current project characteristics.

COCOMO was first published in 1981 Barry W. Boehm's Book Software engineering economics [1] as a model for estimating effort, cost, and schedule for software projects. It drew on a study of 63 projects at TRW Aerospace where Barry Boehm was Director of Software Research and Technology in 1981. The study examined projects ranging in size from 2,000 to 100,000 lines of code, and programming languages ranging from assembly to PL/I. These projects were based on the waterfall model of software development which was the prevalent software development process in 1981.

### Project Classification:

- Organic projects - "small" teams with "good" experience working with "less than rigid" requirements. (2-50 KLOC)
- Semi-detached projects - "medium" teams with mixed experience working with a mix of rigid and less than rigid requirements. (50-300KLOC)
- Embedded projects - developed within a set of "tight" constraints (hardware, software, operational, ..) (>300 KLOC)

## Software Estimation Can be done by three categories or Forms of COCOMO Calculation:

### 1. Basic COCOMO:

By means of parametric estimation equations (differentiated according to the different system types) the development effort and the development duration are calculated on the basis of the estimated DSI.

The breakdown to phases is realized in percentages. In this connection it is differentiated according to system types (organic-batch, semidetached-on-line, embedded-real-time) and project sizes (small, intermediate, medium, large, very large).

### 2. Intermediate COCOMO:

The estimation equations are now taking into consideration (apart from DSI) 15 influence factors; these are product attributes (like software reliability, size of the database, complexity), computer attributes (like computing time restriction, main memory restriction), personnel attributes (like programming and application experience, knowledge of the programming language), and project attributes (like software development environment, pressure of development time). The degree of influence can be classified as very low, low, normal, high, very high, extra high; the multipliers can be read from the available tables.

### 3. Detailed COCOMO:

In this case the breakdown to phases is not realized in percentages but by means of influence factors allocated to the phases. At the same time, it is differentiated according to the three levels of the product hierarchy (module, subsystem, system); product-related influence factors are now taken into consideration in the corresponding estimation equations.

How to determine the estimation parameter:

Effort= $a*(size)^b$

Where, a and b are constants

Size must be in KLOC

Effort → Person – Months (PM)







### 1. Estimation of development Effort

a. Organic Project:

Effort(E)= $2.4*(size)^{1.05}$ PM

b. Semi-detached Project:

Effort(E)= $3*(size)^{1.12}$ PM

c. Embedded Project:

Effort(E)= $3.6*(size)^{1.20}$ PM

### 2. Estimation of development Time:

a) Organic Project:

$T_{Dev}$= $2.5*(effort)^{0.38}$ Months

b) Semi-detached Project:

$T_{Dev}$= $2.5*(effort)^{0.35}$ Months

c) Embedded Project:

$T_{Dev}$= $2.5*(effort)^{0.32}$ Months

### 3. Average Staff Size:

Average Staff Size (SS): $\frac{Effort}{T_{Dev}}$ Persons



### 4. Productivity:

Productivity (P): $\frac{Size}{Effort}$ KLOC/PM

**Question 1**: A project size of 200 KLOC is to be developed. Software development team has average experience on similar types of projects. The schedule is not very tight. Calculate the effort development time, Average staff size and the productivity of the project.

Ans: sol$^n$,

Size = 200 KLOC

According to the given parameter it is semi-detached project.

Effort(E)= 3 *(size)$^{1.12}$ PM

= 3* ( 200)$^{1.12}$ PM

= 1133.117 PM

Development Time ($T_{Dev}$) = 2.5* (Effort)$^{0.35}$ Months

= 2.5* (1133.117)$^{0.35}$ Months

= 29.3 Months

Average Staff Size (SS) = $\frac{Effort}{T_{Tdev}}$ Persons

= $\frac{1133.117}{29.3}$ Persons

= 38.6 Persons

Productivity (P) = $\frac{Size}{Effort}$ KLOC/PM

= $\frac{200}{1133.117}$ KLOC/PM

= 0.1765 LOC

= 0.1765*1000 KLOC/PM

= 17.65 KLOC/PM





**Question 2**: A development project is estimated to be about 55 KLOC when complete and is believed to be medium complexity. It will be a web enabled system with a robust backend database. Calculate effort, development time, productivity and average staff size.

Ans: sol$^n$,

Size = 55 KLOC

According to the given parameter it is semi-detached project.

Effort(E)= 3 *(size)$^{1.12}$ PM

= 3* (55)$^{1.12}$ PM

= PM

Development Time ($T_{Dev}$) = 2.5* (Effort)$^{0.35}$ Months

= 2.5* ( )$^{0.35}$ Months

= Months

Average Staff Size (SS) = $\frac{Effort}{T_{Tdev}}$ Persons

= – Persons

= Persons

Productivity (P) = $\frac{Size}{Effort}$ KLOC/PM

= $\frac{55}{}$ KLOC/PM

= LOC

= *1000 KLOC/PM

= KLOC/PM

## Risk Management

[ ➔ was discussed during the class]

## Staffing

[ ➔ was discussed during the class]

** End of Chapter 3 **







# Chapter: 4
# Software Requirement and Specification

## Types of requirement

1. User requirements

Statements in natural language plus diagrams of the services the system provides and its operational constraints. Written for customers

2. System requirements

A structured document setting out detailed descriptions of the system services. Written as a contract between client and contractor

3. Software specification

A detailed software description which can serve as a basis for a design or implementation. Written for developers

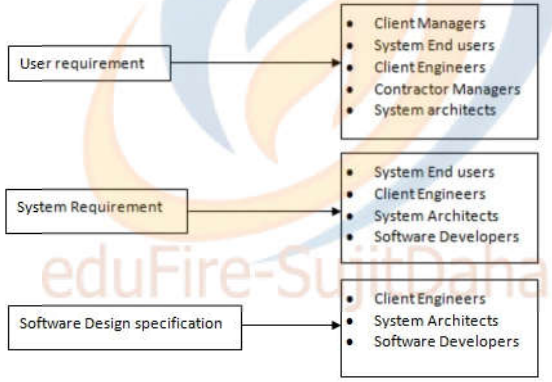



**Fig. 3.1 Readers of Different Types of Specifications**

## Functional and non-functional requirements

Functional Requirement:

These are statements of services the system should provide, how the system should react to particular inputs and how the system should behave in particular situations. In some cases, functional requirements may also explicitly state what the system should not do. Requirements, which are related to functional aspect of software fall into this category. They define functions and functionality within and from the software system.

Functional requirements describe what the system should do. The functional requirement is describing the behavior of the system as it relates to the system's functionality. Statements of services the system should provide how the system should react to particular inputs and how the system should behave in particular situations. In some cases, the functional requirements may also explicitly state what the system should no do.

EXAMPLES -

- Search option given to user to search from various invoices.
- User should be able to mail any report to management.
- Users can be divided into groups and groups can be given separate rights.
- Should comply business rules and administrative functions.
- Software is developed keeping downward compatibility intact.

Non-Functional Requirements:

These are constraints on the services or functions offered by the system. They include timing constraints, constraints on the development process, standards and so on. The functional requirements definition of a system should be both complete and consistent. Completeness means that all services required by the user should be defined. Consistency means that requirements should not have contradictory definitions. There are three types of major problem with requirements definitions written in natural languages:

- **Lack of clarity:** It is very difficult to use language in a precise and unambiguous way without making the document wordy and difficult to read.
- **Requirements confusion:** Functional, non-functional requirements, system goals and design information may not be clearly distinguished.
- **Requirements amalgamation:** Several different requirements may be expressed together as a single requirement

Some organizations try to produce a single specification to act as both a requirements definition and a requirements specification. When a requirements definition is combined with a specification there is often confusion between concepts and details. Requirements, which are not related to functional aspect of software, fall into this category. They are implicit or expected characteristics





of software, which users make assumption of. **Non-functional requirements describe how the system works.** The non-functional requirement **elaborates a performance characteristic** of the system. These are constraints on the services or functions offered by the system such as timing constraints, constraints on the development process, standards, etc. Non-functional requirements often apply to the system as a whole. They do not usually just apply to individual system features or services.

Non-functional requirements include:

- Security
- Storage
- Configuration
- Performance
- Cost
- Interoperability
- Flexibility
- Disaster recovery
- Accessibility

Difference between function and non-function requirement

| Parameters | Functional Requirement | Non-Functional Requirement |
|---|---|---|
| What it is | Verb | Attributes |
| Requirement | It is mandatory | It is non-mandatory |
| Capturing type | It is captured in use case. | It is captured as a quality attribute. |
| End-result | Product feature | Product properties |
| Capturing | Easy to capture | Hard to capture |
| Objective | Helps you verify the functionality of the software. | Helps you to verify the performance of the software. |
| Area of focus | Focus on user requirement | Concentrates on the user's expectation. |
| Documentation | Describe what the product does | Describes how the product works |
| | | |
| Test Execution | Test Execution is done before non-functional testing. | After the functional testing |
| Product Info | Product Features | Product Properties |

## Requirements Engineering Process:

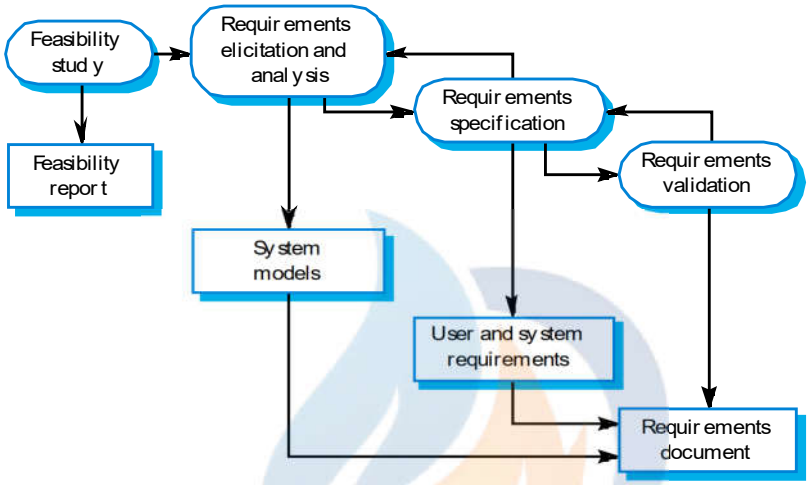



Fig: Requirement Engineering Process

The process to gather the software requirements from client, analyze and document them is known as requirement engineering. The goal of requirement engineering is to develop and maintain sophisticated and descriptive 'System Requirements Specification' document. It is a five steps process, which includes –

- Feasibility Study
- Requirement Elicitation and Analysis
- SRS
- Software Requirement Validation
- Requirement Management

Let us see the process briefly -

### 1. Feasibility study

For all system, the requirements engineering process should start with a feasibility study. The input to the feasibility study is an outline description of the system and how it will be used within an organization. The result of feasibility study should be a report which recommends whether or not





it is worth carrying on with the requirements engineering and system development process. A feasibility study focused to study the following questions:

- Does the system contribute to the overall objectives of the organization?
- Can the system be implemented using current technology and within given cost and schedule constraints?
- Can the system be integrated with other systems which are already in place?
- Carrying out a feasibility study involves information assessment, information collection and report writing.
- The information assessment phase identifies the information which is required to answer the three questions set as above.
- When the information is available, the feasibility study report is prepared.

This should make a recommendation about whether or not the system development should continue. When the client approaches the organization for getting the desired product developed, it comes up with rough idea about what all functions the software must perform and which all features are expected from the software. Referencing to this information, the analysts does a detailed study about whether the desired system and its functionality are feasible to develop.

This feasibility study is focused towards goal of the organization. This study analyzes whether the software product can be practically materialized in terms of implementation, contribution of project to organization, cost constraints and as per values and objectives of the organization. It explores technical aspects of the project and product such as usability, maintainability, and productivity and integration ability.

The output of this phase should be a feasibility study report that should contain adequate comments and recommendations for management about whether or not the project should be undertaken.

### 2. Requirement Gathering (Elicitation) and Analysis:

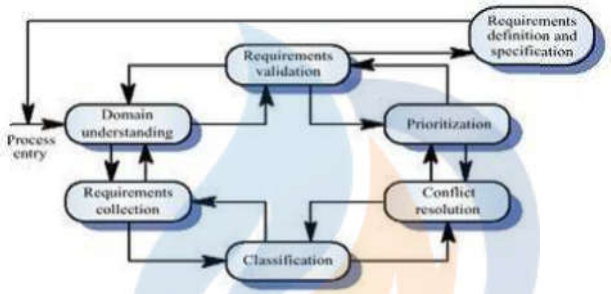
The second step in requirement engineering is requirements elicitation and analysis. In this activity, technical software development staffs work with customers and system end-users to find out about the application domain, what services the system should provide, the required performance of the system, hardware constraints, and so on.

May involve a variety of different kinds of people in a organization. The term stakeholder is used to refer to anyone who should have some direct or indirect influence on the system requirements. Elicitation and analysis is a difficult process for a number of reasons:

- Stakeholders often don't really know what they want from the computer system except in the most general terms; they may find it difficult to articulate what they want from the system.

- Stakeholders in a system naturally express requirements in their own terms and with implicit knowledge of their own work. Requirements engineers, without experience in the customer's domain, must understand these requirements.
- Different stakeholders have different requirements and they may express this in different ways.
- Political factors may influence the requirements of the system.
- The economic and business environment in which the analysis takes place is dynamic.

- A generic process model of the elicitation and analysis process is shown in the figure below:




The process activities are:

- Domain Understanding. Analysts must develop their understanding of the application domain.
- Requirements collection. This is the process of interacting with stakeholders in the system to discover their requirements.
- Classification. This activity takes the unstructured collection of requirements and organizes them into coherent clusters.
- Conflict resolution. Where multiple stakeholders are involved, requirements will surely conflict. This activity is concerned with finding and resolving these conflicts.
- Prioritization. This stage involves interaction with stakeholders to discover the most important requirements.
- Requirements checking. This stage involves o checking of requirements, if they are complete, consistent and in accordance with what stakeholders really want from the system.

If the feasibility report is positive towards undertaking the project, next phase starts with gathering requirements from the user. Analysts and engineers communicate with the client and end-users to know their ideas on what the software should provide and which features they want the software to include.





**Requirements elicitation** for a product line must capture anticipated variations explicitly over the for useable lifetime of the product line. This means that the community of stakeholders is probably larger than for single-system requirements elicitation and may well include domain experts, market experts, and others. Requirements elicitation focuses on the scope, explicitly capturing the anticipated variation by the application of domain analysis techniques, the incorporation of existing domain analysis models, and the incorporation of use cases that capture the variations that are expected to occur over the lifetime of the product line (e.g., change cases and use case variation points.

## 3. SRS

[➔discussed during the class]

## 4. Software Requirement Validation

It is concerned with showing that the requirements actually define the system which the customer wants. It is important because errors in a requirements document can lead to extensive rework costs when they are subsequently discovered during development or after the system is in service. During the requirements validation process, different types of checks should be carried out on the requirements in the requirements document. These checks include:

a. **Validity Checks:** A user may think that a system is needed to perform certain functions. Systems have diverse users with different needs and any set of requirements is inevitably a compromise across the user community.
b. **Consistency Checks:** Requirements in the document should not conflict. That is, there should not be contradictory or different descriptions of the same system function.
c. **Completeness Checks:** The requirements document should include requirements which define all functions and constraints intended by the system user.
d. **Realism Checks:** Using knowledge of existing technology, the requirements should be checked to ensure that they can actually be implemented. These checks also take account of the budget and schedule for the system development.
e. **Verifiability:** To reduce the potential for dispute between customer and contractor, system requirements should always be written so that they are verifiable.

There are number of requirements validation techniques which can be used in conjunction or individually:

a. **Requirements reviews**. The requirements are analyzed systematically by the team of reviewers.

b. **Prototyping**. In this approach to validation, an executable model of the system is demonstrated to end users and customers. They can experiment with this model to see if it meets their real needs.

c. **Test-case generation**. Requirements should, ideally, be testable. If a test is difficult or impossible to design, this usually means that the requirements will be difficult to implement and should b reconsidered.

d. **Automated consistency analysis**. If the requirements are expressed as a system model in a structured or formal notation then CASE tools may be used to check the consistency of the model

After requirement specifications are developed, the requirements mentioned in this document are validated. User might ask for illegal, impractical solution or experts may interpret the requirements incorrectly. This results in huge increase in cost if not nipped in the bud. Requirements can be checked against following conditions -

- If they can be practically implemented
- If they are valid and as per functionality and domain of software
- If there are any ambiguities
- If they are complete
- If they can be demonstrated

## 5. Requirement Management:

The requirements for large software systems are always changing. Because the problem cannot be fully defined, the software requirements are bound to be incomplete. During the software process, the developer's understanding of the problem is constantly changing and these changes feed back to the requirements. Requirement management is the process of understanding and controlling changes to system requirements. The process of requirements management is carried out in conjunction with other requirements engineering processes.

**Requirements management** must now make allowances for the dual nature of the requirements engineering process and the staged (common, specific) nature of the activity. Change-management policies must provide a formal mechanism for proposing changes in the product line and supporting the systematic assessment of how the proposed changes will impact the product line. Change-management policies govern how changes in the product line requirements are proposed, analyzed, and reviewed. The coupling between the product line requirements and the core assets is leveraged by the use of traceability links between those requirements and their associated core assets. Changes in the requirements can then trigger the appropriate changes in the related core assets.





## 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

a) 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.

b) Surveys:

Organization may conduct surveys among various stakeholders by querying about their expectation and requirements from the upcoming system.

c) Questionnaires:

A document with pre-defined set of objective questions and respective options is handed over to all stakeholders to answer, which are collected and compiled.

A shortcoming of this technique is, if an option for some issue is not mentioned in the questionnaire, the issue might be left unattended.

d) Task analysis:

Team of engineers and developers may analyze the operation for which the new system is required. If the client already has some software to perform certain operation, it is studied and requirements of proposed system are collected.

e) Domain Analysis:

Every software falls into some domain category. The expert people in the domain can be a great help to analyze general and specific requirements.

f) Brainstorming:

An informal debate is held among various stakeholders and all their inputs are recorded for further requirements analysis.

g) Prototyping:

Prototyping is building user interface without adding detail functionality for user to interpret the features of intended software product. It helps giving better idea of requirements. If there is no software installed at client's end for developer's reference and the client is not aware of its own requirements, the developer creates a prototype based on initially mentioned requirements. The prototype is shown to the client and the feedback is noted. The client feedback serves as an input for requirement gathering.

h) Observation:

Team of experts visits the client's organization or workplace. They observe the actual working of the existing installed systems. They observe the workflow at client's end and how execution problems are dealt. The team itself draws some conclusions which aid to form requirements expected from the software.







## Software Prototyping (Prototyping Techniques):

### 1) Throwaway Prototyping:

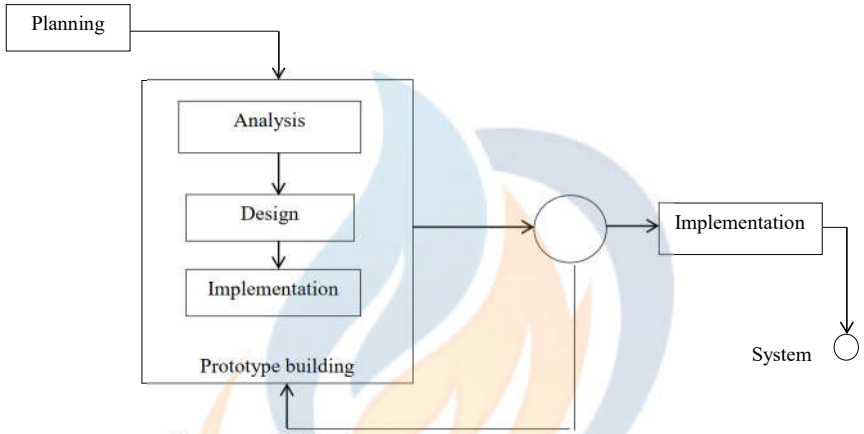



Fig: Throwaway Prototyping

- This approach extends the requirements analysis process with the intention of reducing overall life-cycle costs.
- The principal function of the prototype is to classify requirements and provide additional information for managers to assess process risk.
- After the evaluation, the prototype is thrown away.
- This approach is commonly used for hardware systems.
- Throw-away software prototype is not normally used for design validation but to help develop the system requirements.
- There are several problems to this approach:
    1. Important features may have been left out of the prototype to simplify rapid implementation.
    2. An implementation has no legal standing as a contract between customer and contractor.
    3. Non-functional requirements such as those concerning reliability, robustness and safety cannot be adequately tested in a prototype implementation.
- Throw-away prototypes do not have to be executable software prototypes to be useful in the requirements engineering process.

With 'throw-away' prototyping a small part of the system is developed and then given to the end user to try out and evaluate. The user provides feedback which can quickly be incorporated into the development of the main system.

The prototype is then discarded or thrown away. This is very different to the evolutionary approach.

The objective of throw-away prototyping is to ensure that the system requirements are validated and that they are clearly understood.

The throw-away prototype is NOT considered part of the final system. It is simply there to aid understanding and reduce the risk of poorly defined requirements. The full system is being developed alongside the prototypes and incorporates the changes needed.

The advantage of this approach is the speed with which the prototype is put together. It also focuses the user on only one aspect of the system so keeping their feedback precise.

One disadvantage with throw-away prototyping is that developers may be pressurised by the users to deliver it as a final system! Another issue is that all the man-hours of putting together the throw away prototypes are lost unlike the evolutionary approach. But the benefits may outweigh the disadvantages.

Which approach to use (evolutionary or throw-away) will depend on the nature of the system being developed?

- After preliminary requirement gathering is accomplished, a simple working model of the system is constructed to visually shows the users what their requirements may look like when they are implemented into finished system.
- This prototyping involves creating a working model of various parts of the system at a very early stage, after a relatively short investigation.
- The model then becomes a starting point from which users can re-examine their expectation and clarify their requirements. When this has been achieved the prototype model is thrown away and the system is formally developed based on the identified requirement.
- The reason for using throwaway prototyping is that it can be done quickly. If the user can get week feedback on their requirements' they may be able to refine them early in the development of the software.
- Another strength of throwaway prototyping is its ability to construct interfaces that the user can test.





## 2) Evolutionary Prototyping:

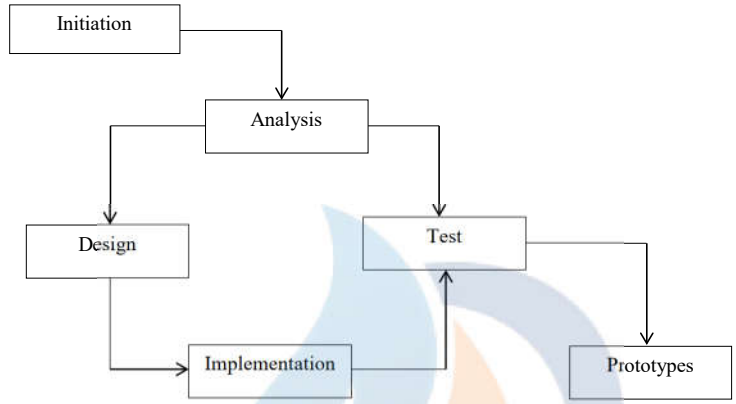



Fig: Evolutionary Prototyping model

- Based on the idea of developing an initial implementation, exposing this to user comment and refining this through many stages until an adequate system has been developed.
- This approach to development was used initially for those systems which are difficult or impossible to specify.
- There are two main advantages to adopting this approach to software development:
    1. Accelerated delivery of the system.
    2. User engagement with the system.

Evolutionary prototyping requirements are gathered in this phase. This evolutionary prototyping models is the main focus of the project managers and stake holders. Meeting with managers, stake holders and users are held in order to determine the requirements. Who is going to use the system? How will they use the system? What date should be input into the system.

- The main goal of using evolutionary prototype is to build a very robust prototype in a structure manner and constantly refine it.
- Using evolutionary prototyping the system is continually refined and re-built.
- Evolutionary prototyping acknowledges that we do not understand all the requirements and builds only those they are well understood. This technique allows the development team to add features or make changes that couldn't be conceived during the requirements and design phase.

### 3) Incremental Prototyping:

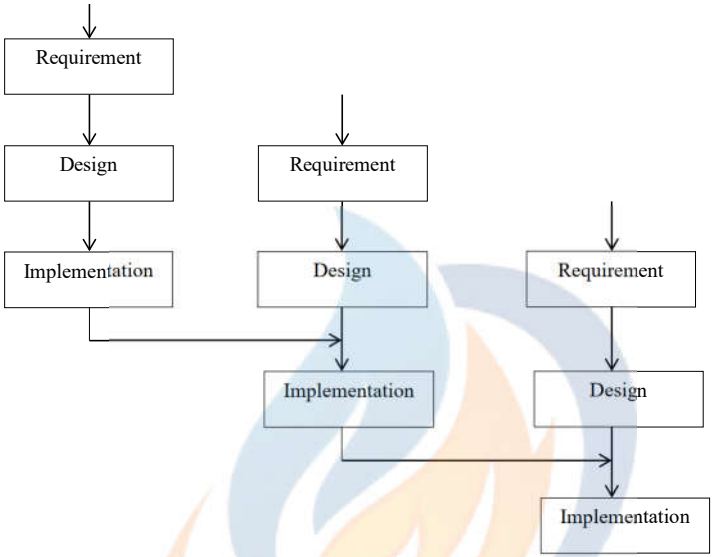



**Fig: Incremental Prototyping**

The final product is built as separate prototypes. At the end the separate prototype are being merged in an overall design.







## Data Modeling and Flow Diagram

### Data Modeling

**Data modeling (data modelling)** is the process of creating a data model for the data to be stored in a Database. This data model is a conceptual representation of Data objects, the associations between different data objects and the rules. Data modeling helps in the visual representation of data and enforces business rules, regulatory compliances, and government policies on the data. Data Models ensure consistency in naming conventions, default values, semantics, security while ensuring quality of the data.

Data model emphasizes on what data is needed and how it should be organized instead of what operations need to be performed on the data. Data Model is like architect's building plan which helps to build a conceptual model and set the relationship between data items.

The two types of Data Models techniques are:

- Entity Relationship (E-R) Model
- UML (Unified Modelling Language)

Types of Data Models

There are mainly three different types of data models:

1. **Conceptual:** This Data Model defines **WHAT** the system contains. This model is typically created by Business stakeholders and Data Architects. The purpose is to organize, scope and define business concepts and rules.
2. **Logical:** Defines **HOW** the system should be implemented regardless of the DBMS. This model is typically created by Data Architects and Business Analysts. The purpose is to developed technical map of rules and data structures.
3. **Physical**: This Data Model describes **HOW** the system will be implemented using a specific DBMS system. This model is typically created by DBA and developers. The purpose is actual implementation of the database.

1. Conceptual Model

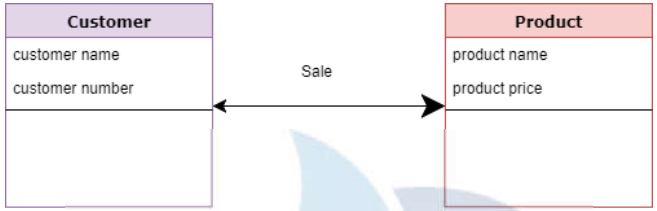
The main aim of this model is to establish the entities, their attributes, and their relationships. In this Data modeling level, there is hardly any detail available of the actual Database structure.

The 3 basic tenants of Data Model are:

- Entity: A real-world thing
- Attribute: Characteristics or properties of an entity
- Relationship: Dependency or association between two entities

For example:

- Customer and Product are two entities. Customer number and name are attributes of the Customer entity
- Product name and price are attributes of product entity
- Sale is the relationship between the customer and product




## Characteristics of a conceptual data model

- Offers Organisation-wide coverage of the business concepts.
- This type of Data Models are designed and developed for a business audience.
- The conceptual model is developed independently of hardware specifications like data storage capacity, location or software specifications like DBMS vendor and technology. The focus is to represent data as a user will see it in the "real world."

Conceptual data models known as Domain models create a common vocabulary for all stakeholders by establishing basic concepts and scope.

2. Logical Data Model

Logical data models add further information to the conceptual model elements. It defines the structure of the data elements and set the relationships between them.


The advantage of the Logical data model is to provide a foundation to form the base for the Physical model. However, the modeling structure remains generic.





At this Data Modeling level, no primary or secondary key is defined. At this Data modeling level, you need to verify and adjust the connector details that were set earlier for relationships.

## Characteristics of a Logical data model

- Describes data needs for a single project but could integrate with other logical data models based on the scope of the project.
- Designed and developed independently from the DBMS.
- Data attributes will have datatypes with exact precisions and length.
- Normalization processes to the model is applied typically till 3NF.

3. Physical Data Model

A Physical Data Model describes the database specific implementation of the data model. It offers an abstraction of the database and helps generate schema. This is because of the richness of meta-data offered by a Physical Data Model.


This type of Data model also helps to visualize database structure. It helps to model database columns keys, constraints, indexes, triggers, and other RDBMS features.

[➔was discussed in more detail during the class]

### Process Modeling

Where the process from one state to another state is described. How data is processed by system in terms of input and output is explained. i.e. where data comes from, where it goes and how it gets stored. For eg: DFD

[➔was discussed in more detail during the class]

## Requirement Definition and Specification

[➔was discussed during the class]

# Chapter: 5
# Software Design

## Introduction to software Design

Software design is a process to transform user requirements into some suitable form, which helps the programmer in software coding and implementation.

For assessing user requirements, an SRS (Software Requirement Specification) document is created whereas for coding and implementation, there is a need of more specific and detailed requirements in software terms. The output of this process can directly be used into implementation in programming languages.

The design phase of software development deals with transforming the customer requirements as described in the SRS documents into a form implementable using a programming language.

Software design is the first step in SDLC (Software Design Life Cycle), which moves the concentration from problem domain to solution domain. It tries to specify how to fulfill the requirements mentioned in SRS.

- Software design sits at the technical kernel of software engineering and is applied regardless of the software process model that is used.
- Software design is the first of three technical activities: design, code generation, and test – that are required to build and verify the software.
- Using one of the numbers of design methods, the design task produce a data design, an architectural design, an interface design, and a component design.
- The data design transforms the information domain model created during analysis into the data structures that will be required to implement the software.
- The architectural design defines the relationship between major structural elements of the software.
- The interface design describes how the software communicates within itself, with systems that incorporate with it and with humans who use it.
- The component level design transforms structural elements of the software architecture into a procedural description of the software components.
- The importance of software design can be state with a single word: **"QUALITY"**.

## Characteristics of good software design

[➔was discussed during the class]





## Design Principles

Software design is both a process and a model. The design process is a sequence of steps that enable the designer to describe all aspects of the software to be built. The design model is equivalent of an architect's plans for a house. The design model that is created for software provides a variety of different views of the computer software. Basic design principles enable the software engineer to navigate the design process. Davis suggests a set of principles for the software design:

1. **The design process should not suffer from "tunnel vision."** A good designer should consider alternative approaches, judging each based on the requirements of the problem, the resources available to do the job.
2. **The design should be traceable to the analysis model.** Because a single element of the design model often traces to multiple requirements, it is necessary to have a means for tracking how requirements have been satisfied by the design model.
3. **The design should not reinvent the wheel.** Systems are constructed using a set of design patterns, many of which have likely been encountered before. These patterns should always be chosen as an alternative to reinvention. Time is short and resources are limited! Design time should be invested in representing truly new ideas and integrating those patterns that already exist.
4. **The design should "minimize the intellectual distance" between the software and the problem as it exists in the real world.** That is, the structure of the software design should (whenever possible) mimic the structure of the problem domain.
5. **The design should exhibit uniformity and integration.** A design is uniform if it appears that one person developed the entire thing. Rules of style and format should be defined for a design team before design work begins. A design is integrated if care is taken in defining interfaces between design components.
6. **The design should be structured to accommodate change.** The design concepts discussed in the next section enable a design to achieve this principle.
7. **The design should be structured to degrade gently, even when aberrant data, events, or operating conditions are encountered.** Well- designed software should never ―bomb.‖ It should be designed to accommodate unusual circumstances, and if it must terminate processing, do so in a graceful manner.
8. **Design is not coding, coding is not design.** Even when detailed procedural designs are created for program components, the level of abstraction of the design model is higher than source code. The only design decisions made at the coding level address the small implementation details that enable the procedural design to be coded.
9. **The design should be assessed for quality as it is being created, not after the fact.** A variety of design concepts and design measures are available to assist the designer in assessing quality.
10. **The design should be reviewed to minimize conceptual (semantic) errors.** There is sometimes a tendency to focus on minutiae when the design is reviewed, missing the forest

for the trees. A design team should ensure that major conceptual elements of the design (omissions, ambiguity, inconsistency) have been addressed before worrying about the syntax of the design model.

When the design principles are properly applied, the software engineer creates a design that exhibits both external and internal quality.

Note that design principles are often constrained by the existing hardware configuration, the implementation language, the existing file and data structures, and the existing organizational practices. Also, the evolution of each software design should be meticulously designed for future evaluations, references and maintenance.

## Design Concepts

A set of fundamental software design concepts has evolved over the past four decades. Each provides the software designer with a foundation from which more sophisticated design methods can be applied. Each helps the software engineer to answer the following questions:

- What criteria can be used to partition software into individual components?
- How function or data is structure detail separated from a conceptual representation to the software?
- What uniform criteria define the technical quality of a software design?

The beginning of wisdom for a software engineer is to recognize the difference between getting a program to work, and getting it right. Every software process is characterized by basic concepts along with certain practices or methods. Methods represent the manner through which the concepts are applied. As new technology replaces older technology, many changes occur in the methods that are used to apply the concepts for the development of software. However, the fundamental concepts underlining the software design process remain the same, some of which are described here.

### 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 is achieved by dividing the software into uniquely named and addressable components, which are also known as **modules.** A complex system (large program) is partitioned into a set of discrete modules in such a way that each module can be developed independent of other modules. After developing the modules, they are integrated together to meet the software requirements. Note that larger the number of modules a system is divided into, greater will be the effort required to integrate the modules.

Modularizing a design helps to plan the development in a more effective manner, accommodate changes easily, conduct testing and debugging effectively and efficiently, and conduct maintenance work without adversely affecting the functioning of the software.

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.

### Stepwise Refinement

Stepwise refinement is a top-down design strategy used for decomposing a system from a high level of abstraction into a more detailed level (lower level) of abstraction. At the highest level of abstraction, function or information is defined conceptually without providing any information about the internal workings of the function or internal structure of the data. As we proceed towards the lower levels of abstraction, more and more details are available.

Software designers start the stepwise refinement process by creating a sequence of compositions for the system being designed. Each composition is more detailed than the previous one and contains more components and interactions. The earlier compositions represent the significant interactions within the system, while the later compositions show in detail how these interactions are achieved.

### 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.

The use of information hiding as design criteria for modular system provides the most significant benefits when modifications are required during testing's and later during software maintenance. This is because as most data and procedures are hidden from other parts of the software,

inadvertent errors introduced during modifications are less likely to propagate to different locations within the software.

Modules should be specified and designed in such a way that the data structures and processing details of one module are not accessible to other modules. They pass only that much information to each other, which is required to accomplish the software functions. The way of hiding unnecessary details is referred to as **information hiding. IEEE** defines information hiding as 'the technique of encapsulating software design decisions in modules in such a way that the module's interfaces reveal as little as possible about the module's inner workings; thus each module is a 'black box' to the other modules in the system.

### Architecture

Software architecture refers to the structure of the system, which is composed of various components of a program/ system, the attributes (properties) of those components and the relationship amongst them. The software architecture enables the software engineers to analyze the software design efficiently. In addition, it also helps them in decision-making and handling risks. The software architecture does the following.

- Provides an insight to all the interested stakeholders that enable them to communicate with each other
- Highlights early design decisions, which have great impact on the software engineering activities (like coding and testing) that follow the design phase
- Creates intellectual models of how the system is organized into components and how these components interact with each other.

### Patterns

A pattern provides a description of the solution to a recurring design problem of some specific domain in such a way that the solution can be used again and again. The objective of each pattern is to provide an insight to a designer who can determine the following.

- Whether the pattern can be reused
- Whether the pattern is applicable to the current project
- Whether the pattern can be used to develop a similar but functionally or structurally different design pattern.





## Software Design Process and Design Quality

**[ ➔ related figure for the process was discussed during the class]**

- Software design is an iterative process through which requirements are translated into a blue-print for construction the software.
- As design iterations occur, subsequent refinements leads to design representations at much lower level of abstractions.
- Throughout the design process, the quality of evolving design is assessed with a series of formal technical reviews or design walkthroughs.
- McGlaugwin suggests three characteristics that serve as a guide for the evaluation of a good design.
  - The design must implement all of the explicit requirements contained in the analysis model, and it must accommodate all of the implicit requirements desired by the customers.
  - The design must be readable, understandable guide for those generate code and for those who test and subsequently support the software.
  - The design should provide a complete picture of software, addressing the data, functional and behavioral domains from an implementation perspective.
- Each of these characteristics is actually a goal of the design process.
- The goal of the design process is achieved by following guidelines:

  - A design should exhibit an architectural design that –
  - A design should be modular, that is, the software should be logically partitioned into elements that perform specific functions and sub-functions.
  - A design should contain distinct representations of data, architecture, interfaces, and components.
  - A design should lead to data structures that are appropriate for the objects to be implemented and are drawn from recognizable data patterns.
  - A design should lead to components that exhibit independent functional characteristics.
  - A design should lead to interfaces that reduce the complexity of connections between modules and with the external environment.
  - A design should be derived using a repeatable method that is driven by information obtained during software requirement analysis.

# Software Architecture and its types

- Software is divided into modules first.
- Software systems have had architectures, and programmers have been responsible for the interactions among the modules and global properties of the assemblage.
- Today, effective software architecture and its explicit representation and design have become dominant themes in software engineering.

## What is Architecture?

- According to Bass, Clements, and Kazman "The software architecture of a program or computing system is the structure or structures of the system, which comprise software components, the externally visible properties of those components, and the relationships among them."
- The architecture is not the operational software.
- It is a representation that enables a software engineer to:
  - analyze the effectiveness of the design in meeting its stated requirements,
  - consider architectural alternatives at a stage when making design changes is still relatively easy, and
  - reducing the risks associated with the construction of the software.
- The software architecture considers two levels of the design pyramid – data design and architectural design.
- Data design enables us to represent the data component of the architecture.
- Architectural design focuses on the representation of the structure of software components, their properties and interactions.

## Why is Architecture important?

- The three key reasons why software architecture is important are:
  - Representations of software architecture are an enable for communication between all parties (stakeholders) interested in the development of a computer-based system.
  - The architectural highlights early decisions that will have a profound impact on all software engineering work that follows and, as important, on the ultimate success of the system as an operational entity.
  - Architecture "constitutes a relatively small, intellectually graspable model of how the system is structured and how its components work together."
- The architectural model provides a Gestalt view of a system, allowing the software engineer to examine is as a whole.
- The architectural design model and the architectural patterns contained within it are transferable.

## Architectural styles

- The software that is built for computer-based systems exhibits one of many architectural styles:





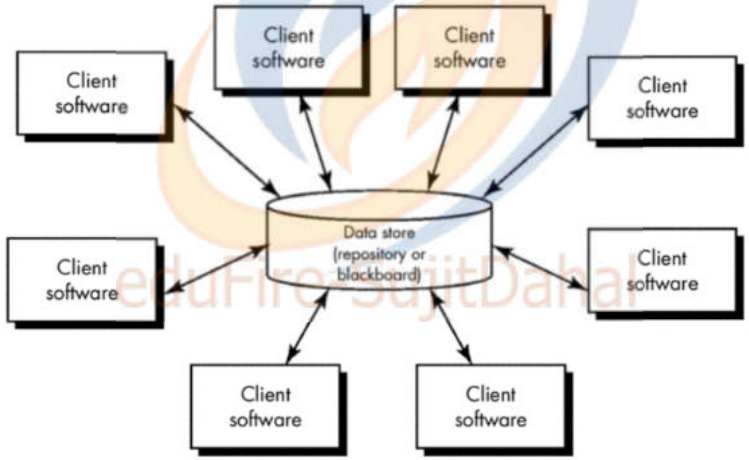
- Each style describes a system category that encompasses:
  - a set of components that perform a function required by a system.
  - a set of connectors that enable "communication, co-ordination, and co-operation" among components.
  - constraints that define how components can integrated to form the system, and
  - semantic models that enable a designer to understand the overall properties of the system by analyzing the known properties of its constituent parts.

### A Brief Taxonomy of Styles and Patterns

- although millions of computer-based systems have been created over the past 50 years, the vast majority can be categorized into one of a relatively small number of architectural styles:

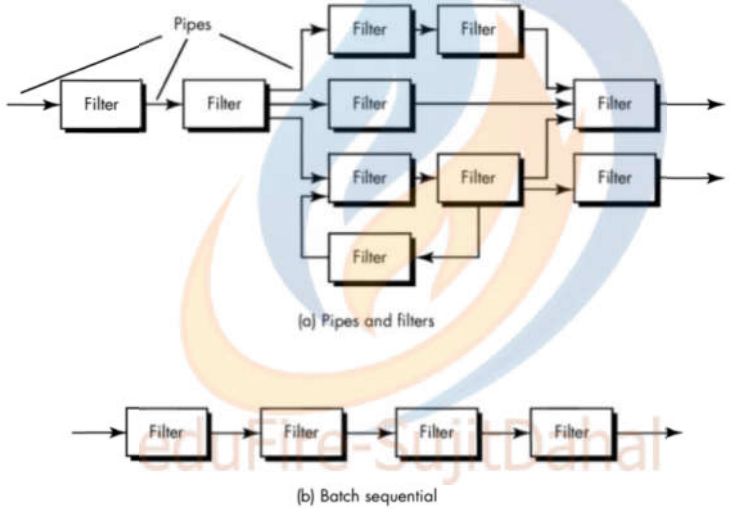
Data-Centered Architecture:

- A data store resides at the center of this architecture and is accessed frequently by other components that update, add, delete, or otherwise modify data within the store.
- Promotes integrability i.e. existing components can be changed and new client components can be added to the architecture without concern about other clients.



Data-Flow Architecture:

- This architecture is applied when input data are to be transformed through a series of computational or manipulative components into output data.
- A pipe and filter pattern has a set of components, called filters, connected by pipes that transmit data from one component to the next.
- Each filter works independently, and does not require knowledge of the working of its neighboring filters.
- If the data flow degenerates into a single line of transforms, it is termed as batch sequential.








Call and Return Architecture:

- This architecture style enables a software designer to achieve a program structure that is relatively easy to modify and scale.
- A number of sub-styles exist within this category:
  - **Main program/sub-program architecture:** This classic program structure decomposes function into a control hierarchy where a "main" program invokes a number of program components, which in turn may invoke other components.
  - **Remote procedure call architecture:** The components of a main-program/sub-program architecture are distributed across a multiple computers in a network.

Object-Oriented Architecture:

- The component of a system encapsulates data and the operations that must be applied to manipulate data.
- Communication and co-ordination between components is accomplished via message passing.

Layered Architecture:

- A number of different layers are defined, each accomplishing operations that progressively become closer to the machine instruction set.
- At the outer layer, components service user interface operations.
- At inner layer, components perform operating system interfacing.
- Intermediate layer provide utility services and application software functions.

## Software Design Strategy

### Structured Design

The problems of the software are categorized into different elements of solution. The design of the solution is considered as the structured design. The way and the process of solving the problem is made easy to understand by structured design. Structured design enables to simplify the problem by the designer. According to the principle of 'divide and conquer', a particular problem is divided into well-structured small problems and each of the small problems is solved.

By using solution modules, the divided problems are solved. The modules are designed in a well organized manner an aims in achieving the best solution for the problem.

The modules are enabled to interact among themselves. Some of the specific rules are followed by the modules for their interaction. They are -

**Cohesion** – All the elements of similar functions are grouped.

**Coupling** – Interaction among the modules.

A structured design that possesses high cohesion and low coupling is considered as a good structured design.

### Function Oriented Design

Functions constitute the sub-systems of a system and which are meant for executing the system tasks. System constitutes the top view for all the functions.

Some of the properties of structured design are followed by function oriented design. Even the Function Oriented Design follows the principle of divide and conquer.

The information and the operation of the transactions are hidden by splitting the system into smaller functions by the function oriented design mechanism. The global information is shared by these functional modules and the information of the modules is shared among themselves.

The state of the program is changed by the function, when that particular function is called by the program. Other modules hesitate the change. Function oriented design best suites where the state of program is not considered and where input is considered by the functions.

### Object Oriented Design

The main focus of object oriented design is on entities. The strategies developed by this object oriented design also focus on the characteristics of entities. There are different concepts used in Object Oriented Design. They are -

**Objects** – Objects are the entities of the system. Examples of objects are company, person, customer etc. Each of the entity has its own attributes which are performed by adopting some methods.

**Classes** – Object is described in a class. The instant of the class is object. The features of the object, functions and methods are described and defined by the class.

Different sources of methods or actions, the functions are described and variables constitute the stored attributes of the solution design.

**Encapsulation** – Encapsulation is a combination of attributes and methods. The data and the methods of the object are not allowed to access by encapsulation. This process of restriction is known as information hiding.

**Inheritance** – Similar classes are piled up in a hierarchy by the process of inheritance. The variables and methods can be imported by the lower sub-class from the next higher class. Inheritance facilitates in creating generalized classes by defining specific classes.





**Polymorphism** – Same name can be assigned to different methods that undertake similar task but differ in arguments. This process in known as polymorphism. A variety of tasks are performed by a single interface. The code is executed on the basis of the function that is invoked.

Design Process

The steps involved in the design process of object oriented design are as follows -

- The earlier system, the system sequence diagram or the requirements can be used for developing a solution design.
- Objects with similar attribute characteristics are identified and grouped into classes.
- Define the relation and the class hierarchy.
- Define the framework of the application.

### Approaches for Software Design?

The following are considered as the approaches for software design.

## Top Down Design

Each system is divided into several sub-systems and components. Each of the sub-systems is further divided into set of sub-systems and components. This process of division facilitates in forming of a system hierarchy structure.

The complete software system is considered as a single entity and in relation to the characteristics; the system is split into sub-system and component. The same is done with each of the sub-systems. This process is continued until the lowest level of the system is reached.

The design is started initially by defining the system as a whole and then keeps on adding definitions of the sub-systems and components. When all the definitions are combined together, it turns out to be a complete system.

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

## Bottom-up Design

The design starts with the lowest level components and sub-systems. Y using these components, the next immediate higher level components and sub-systems are created or composed. The process is continued till all the components and sub-systems are composed into a single component and which is considered as the complete system. The amount of abstraction grows high as the design moves to more high levels.

By using the basic information existing system, when a new system needs to be created, the bottom-up strategy best suits the purpose.

➔It is to be considered that neither of the strategies is practiced individually. They are always combined together and practiced.

[ ➔ definition and term related to strategy were discussed during the class]

***** End of Chapter 5 *****







# Chapter: 6
# Software Testing

## Concept

Software Testing is defined as an activity to check whether the actual results match the expected results and to ensure that the software system is Defect free. It involves execution of a software component or system component to evaluate one or more properties of interest. Software testing also helps to identify errors, gaps or missing requirements in contrary to the actual requirements. It can be either done manually or using automated tools. Some prefer saying Software testing as a White Box and Black Box Testing.

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

Testing is the process of evaluating a system or its component(s) with the intent to find whether it satisfies the specified requirements or not. Testing is executing a system in order to identify any gaps, errors, or missing requirements in contrary to the actual requirements.

## Software Testing Principles:

1. **All tests should be based on customer requirements**
   As the objective of software testing is to overcome errors. It allows that the severe defects (from customer's point of view) are those that cause the program to fail to meet its requirements.

2. **Test should be planned long before testing begins.**
   Test planning can begin as soon as the requirements model is complete.
   Testers need to get involved at an early stage of the Software Development Life Cycle (SDLC). Thus the defects during the requirement analysis phase or any documentation defects can be identified. The cost involved in fixing such defects is very less when compared to those that are found during the later stages of testing.

3. **The Pareto principle applies to software testing. i.e. Defect Clustering**
   The Pareto principle implies that 80% of errors uncovered during testing will likely be traceable to 20% of all program components.

4. **Testing should begin "in the small" and progress toward testing "in the large"**.
   The first tests planned and executed generally focus on individual components. As testing progresses, focus shifts in an attempt to find errors in integrated clusters of components and ultimately in the entire system.

5. **To be most effective, testing should be conducted by an independent third party.**
   By most effective, mean testing that has the highest probability of finding errors. The software engineer who created the system is not the best person to conduct all tests for the software.

6. **Testing Shows *the Presence of Defects***
   Every application or product is released into production after a sufficient amount of testing by different teams or passes through different phases like System Integration Testing, User Acceptance Testing, and Beta Testing etc.
   So *have you ever seen or heard from any of the testing team that they have tested the software fully and there is no defect in the software*? Instead of that, every testing team confirms that the software meets all business requirements and it is functioning as per the needs of the end user.
   In the software testing industry, no one will say that there is **no defect** in the software, which is quite true as testing cannot prove that the software is error-free or defect-free.

7. **Exhaustive Testing is Not Possible**
   It is not possible to test all the functionalities with all valid and invalid combinations of input data during actual testing. Instead of this approach, testing of a few combinations is considered based on priority using different techniques. Exhaustive testing will take unlimited efforts and most of those efforts are ineffective. Also, the project timelines will not allow testing of so many number of combinations. Hence it is recommended to test input data using different methods like Equivalence Partitioning and Boundary Value Analysis.
   For Example, If suppose we have an input field which accepts alphabets, special characters, and numbers from 0 to 1000 only. Imagine how many combinations would appear for testing, it is not possible to test all combinations for each input type. The testing efforts required to test will be huge and it will also impact the project timeline and cost. Hence it is always said that exhaustive testing is practically not possible.
   The number of path permutations for even a moderately sized program is exceptionally large. For this reason, it is impossible to execute every combination of paths during testing.

8. **Testing is Context Dependent:**
   Testing approach depends on the context of the software we develop. We do test the software differently in different contexts. For example, online banking application requires a different approach of testing compared to an e-commerce site.



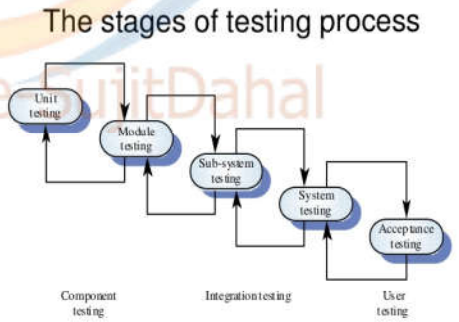


9. **Defect Clustering:**
   Defect Clustering in software testing means that a small module or functionality contains most of the bugs or it has the most operational failures.
   As per the Pareto Principle (80-20 Rule), 80% of issues comes from 20% of modules and remaining 20% of issues from remaining 80% of modules. So we do emphasize testing on the 20% of modules where we face 80% of bugs.

10. **Pesticide Paradox**
    Pesticide Paradox principle says that if the same set of test cases are executed again and again over the period of time then these set of tests are not capable enough to identify new defects in the system. In order to overcome this "Pesticide Paradox", the set of test cases needs to be regularly reviewed and revised. If required a new set of test cases can be added and the existing test cases can be deleted if they are not able to find any more defects from the system.

11. **Absence of Error - Fallacy**
    If the software is tested fully and if no defects are found before release, then we can say that the software is 99% defect free. But what if this software is tested against wrong requirements? In such cases, even finding defects and fixing them on time would not help as testing is performed on wrong requirements which are not as per needs of the end user. For Example, suppose the application is related to an e-commerce site and the requirements against "Shopping Cart or Shopping Basket" functionality which is wrongly interpreted and tested. Here, even finding more defects does not help to move the application into the next phase or in the production environment.

## Software Testing Process:




Five stages of Testing:

1. **Unit Testing:** Individual components are tested to ensure that they operate correctly. Each component is tested independently without other system components.
2. **Module Testing:** This involves the testing of independent components such as procedures and functions. A module encapsulates related components so it can be tested without other system modules.

3. **Subsystem Testing:** This phase involves testing collections of modules which have been integrated into sub-systems. Sub-systems may be independently designed. The most common problems which arise in large software systems are sub-system interface mismatches. The sub-system test process should therefore concentrate on the detection of interface errors by rigorously exercising the interfaces.
4. **System Testing:** Sub systems are integrated to make up the entire system. The testing process is concerned with finding errors that result from unanticipated interactions between sub-systems and system components. It is also concerned with validating that the system meets its functional and non-functional requirements.
5. **Acceptance Testing:** This is the final stage in the testing process before the system is accepted for operational use. The system is tested with data supplied by the system procurer rather than simulated test data. Acceptance testing may reveal errors and omissions in the system requirements definition because the real data exercises the system in different ways from the test data. It may also reveal requirements problems where the system's facilities do not really meet the user's needs or the system's performance is not acceptable.

## Test Case Design

The design of tests for software and other engineered products can be as challenging as the initial design of the product itself. A rich variety of test case design methods have evolved for software. This method provides the developer with a systematic approach of testing. More important, methods provide a mechanism that can help to ensure the components of tests and provide the highest likelihood for uncovering errors in software. Any engineered products can be tested in two ways:

- Knowing the specified function that a product has been designed to perform, tests can be conducted that demonstrate each function is fully operational while at the same time searching for errors in each function.
- Knowing the internal workings of a product, tests can be conducted to ensure that "all gears mesh", that is, internal operations are performed according to specifications and internal components have been adequately exercised.

The first approach is called black box testing, and the second, white box testing. Black box test are used to demonstrate the software functions are operational, that input is properly accepted and output is correctly produced, and that the integrity of external information is maintained. White box test of software is predicated on close examination of procedural detail. White box test can be designed only after a component–level design exists, the logical details of the program must be available.





A test case is a document, which has a set of test data, preconditions, expected results and post conditions, developed for a particular test scenario in order to verify compliance against a specific requirement.

A test case provides the description of inputs and their expected outputs to observe whether the software or a part of the software is working correctly. IEEE defines test case as 'a set of input values, execution preconditions, expected results and execution post conditions, developed for a particular objective or test condition such as to exercise a particular program path or to verify compliance with a specific requirement.' Generally, a test case is associated with details like identifier, name, purpose, required inputs, test conditions, and expected outputs.

perform exhaustive testing, is known as ideal test case. Generally, a test case is unable to perform exhaustive testing; therefore, a test case that gives satisfactory results is selected. In order to select a test case, certain questions should be addressed.

- How to select a test case?
- On what basis are certain elements of program included or excluded from a test case?

To provide an answer to these questions, test selection criterion is used that specifies the conditions to be met by a set of test cases designed for a given program. For example, if the criterion is to exercise all the control statements of a program at least once, then a set of test cases, which meets the specified condition should be selected.

The process of generating test cases helps to identify the problems that exist in the software requirements and design. For generating a test case, firstly the criterion to evaluate a set of test cases is specified and then the set of test cases satisfying that criterion is generated.

A test case is a single executable test which a tester carries out. It guides them through the steps of the test. You can think of a test case as a set of step-by-step instructions to verify something behaves as it is required to behave.

A test case usually contains:

- Title
- Description
- Test steps
- Expected result
- Actual result (once tested)

| Test Case # | Test Case Description | Test Steps | Test Data |
|---|---|---|---|
| 1 | Check response when valid email and password is entered | 1) Enter Email Address | Email: abc@gmail.com |
| | | 2) Enter Password | Password: INf9^Oti7^2h |
| | | 3) Click Sign in | |

## Black-Box Testing:

The technique of testing without having any knowledge of the interior workings of the application is called black-box testing. The tester is oblivious to the system architecture and does not have access to the source code. Typically, while performing a black-box test, a tester will interact with the system's user interface by providing inputs and examining outputs without knowing how and where the inputs are worked upon.

This treats the system as one that cannot be seen in detail. The **structure** of the program is not taken into account. The tests are performed based on what the program **does**. This is sometimes called **Functional Testing**.

The functional requirements have been agreed with the customer. Thus they can see that the program performs as it was requested at the specification stage. It is difficult to know just how much of the program coding has been tested in this case, but it is the kind of testing which is very popular with the customer.

The following table lists the advantages and disadvantages of black-box testing.

| Advantages | Disadvantages |
|---|---|
| • Well suited and efficient for large code segments.<br>• Code access is not required.<br>• Clearly separates user's perspective from the developer's perspective through visibly defined roles.<br>• Large numbers of moderately skilled testers can test the application with no knowledge of implementation, programming language, or operating systems. | • Limited coverage, since only a selected number of test scenarios is actually performed.<br>• Inefficient testing, due to the fact that the tester only has limited knowledge about an application.<br>• Blind coverage, since the tester cannot target specific code segments or error-prone areas.<br>• The test cases are difficult to design. |





### a. Equivalence Class Partitioning

- is a black-box testing method that divides the input domain of a program into classes of data from which the test cases can be derived.
- It strives to define a test case that uncovers classes of errors, thereby reducing the total number of test cases that must be developed.
- Test case design for equivalence partitioning is based on an evaluation of equivalence classes for an input condition.
- An input condition is either a specific numeric value, a range of values, a set of related values, or Boolean condition.
- Equivalence classes may be defined according to the following guidelines:
    1. If an input condition specifies a range, one valid and two invalid equivalence classes are defined.
    2. If an input condition requires a specific value, one valid and two invalid equivalence classes are defined.
    3. If an input condition specifies a member of a set, one valid and one invalid equivalence classes are defined.
    4. If an input condition is Boolean, one valid and one invalid classes are defined.

### b. Boundary Value Analysis

- It is used when a great number of errors tends to occur at the boundaries of input domain rather than in the "center".
- It leads to a selection of test cases that exercise boundary values.
- If a test case design technique that complements equivalence partitioning.
- Leads to the selection of test case at the "edges" of the class.
- Guidelines for boundary value analysis are similar in many respects to those provided for equivalence partitioning:
    1. If an input condition specifies a number of values, test cases should be developed that exercise the minimum and maximum numbers.
    2. If an input condition specifies a range bounded by values a and b, test cases should be designed with values a and b and just above and below a and b.
    3. Apply guidelines 1 and 2 to output conditions.
    4. If internal program data structures have prescribed boundaries, be certain to design a test case to exercise the data structure at its boundary.
- By applying these guidelines, boundary testing will be more complete, thereby having a higher likelihood for error detection.
- For example: if we have an input variable x in the range of 1 to 10 then the boundary values can be 1,0,10 and 11.

## White-Box Testing:

White-box testing is the detailed investigation of internal logic and structure of the code. White-box testing is also called **glass testing** or **open-box testing**. In order to perform **white-box** testing on an application, a tester needs to know the internal workings of the code.

The tester needs to have a look inside the source code and find out which unit/chunk of the code is behaving inappropriately.

This treats the system as one in which the workings can be seen. The structure of the program is taken into account when tests are designed for the program. This is sometimes called **Structural Testing**.

The following table lists the advantages and disadvantages of white-box testing.

| Advantages | Disadvantages |
|---|---|
| • As the tester has knowledge of the source code, it becomes very easy to find out which type of data can help in testing the application effectively.<br>• It helps in optimizing the code.<br>• Extra lines of code can be removed which can bring in hidden defects.<br>• Due to the tester's knowledge about the code, maximum coverage is attained during test scenario writing. | • Due to the fact that a skilled tester is needed to perform white-box testing, the costs are increased.<br>• Sometimes it is impossible to look into every nook and corner to find out hidden errors that may create problems, as many paths will go untested.<br>• It is difficult to maintain white-box testing, as it requires specialized tools like code analyzers and debugging tools. |



**Basic Path Testing**

- Basic path testing is a white box testing technique first proposed by Tom McCabe.
- It enables the test case designer to derive a logical complexity measure of a procedural design and use this measure as a guide for defining a basis set of execution paths.

**Flow Graph Notations:**

- Before basis path method can be introduced, a simple notation for the representation of control flow, called flow graph must be introduced.
- It enables you to trace program paths more readily.





- Have to draw a flow graph when the logical control structure of a module is complex.

**a. Cyclomatic Complexity:**

- It is a software metric that provides a quantitative measure of the logical complexity of a program.
- The value computed for the cyclomatic complexity defines the number of independent paths in the basis set of a program and provides us with an upper-bound for the number of tests that must be conducted to ensure that all statements have been executed at least once.
- An independent path is any path through the program that introduces at least one new set of processing statements or a new condition.
- Cyclomatic complexity is a useful metric for predicting those modules that are likely to be error prone.
- It can be used for test planning as well as test case design.
- Cyclomatic complexity is computed in one of three ways:

1. The number of regions of the flow graph corresponds to the cyclomatic complexity.

2. Cyclomatic complexity, V (G), for a flow graph, G, is defined as:
   $$V(G) = E - N + 2P$$
   Where E is the number of flow graph edges, N is the number of flow graph nodes

3. Cyclomatic complexity, V (G), for a flow graph, G, is also defined as:
   $$V(G) = \pi + 1$$
   Where $\pi$ is the number of predicate nodes contained in the flow graph G.

## b. Statement Coverage:

- It is a measure of percentage of statement that have been executed by test cases.
- The main objective of statement coverage is to achieve 100% statement coverage through testing.



## c. Path Coverage:

In path coverage we write test cases to ensure that each and every path that have been traversed at once. In this testing the path coverage is understand by:

- Drawing the flow graph to indicate corresponding logic in the program.
- By calculating individual paths in the flow graph.
- Running the program in all possible ways to cover every possible statement.

**Difference between Black Box testing and White Box Testing**

| Black Box Testing | White Box Testing |
|---|---|
| 1. Black Box Testing is a software testing method in which the internal structure/ design/ implementation of the item being tested is not known to the tester. Tester is mainly concerned with the validation of output rather than how the output is produced | 1. **White Box Testing** is a software testing method in which the internal structure/ design/ implementation of the item being tested is known to the tester. Tester validates the internal structure of the item under consideration along with the output. |
| 2. Programming knowledge and implementation knowledge (internal structure and working) is not required in Black Box testing. | 2. Programming knowledge and implementation knowledge (internal structure and working) is required in **White Box testing** |
| 3. Black box testing is done by the professional testing team and can be done without knowledge of internal coding of the item. | 3. White Box testing is generally done by the programmers who have developed the item or the programmers who have an understanding of the item's internals. |
| 4. This testing is done by testers. | 4. This testing is mostly done by developers. |
| 5. Black box testing means functional test or external test. | 5. White box testing means structural test or interior test. |
| 6. In this sort of testing testers mainly focuses on the functionality of the system. | 6. In this sort of testing developers mainly focuses on the structure means program/code of the system. |







## Difference between software verification and validation

| Verification | Validation |
|---|---|
| 1. Verification is a static practice of verifying documents, design, code and program. | 1. Validation is a dynamic mechanism of validating and testing the actual product. |
| 2. It does not involve executing the code. | 2. It always involves executing the code. |
| 3. It is human based checking of documents and files. | 3. It is computer based execution of program. |
| 4. Verification uses methods like inspections, reviews, walkthroughs, and Desk-checking etc. | 4. Validation uses methods like black box (functional) testing, gray box testing, and white box (structural) testing etc. |
| 5. **Verification** is to check whether the software conforms to specifications. | 5. **Validation** is to check whether software meets the customer expectations and requirements. |
| 6. It can catch errors that validation cannot catch. It is low level exercise. | 6. It can catch errors that verification cannot catch. It is High Level Exercise. |
| 7. Target is requirements specification, application and software architecture, high level, complete design, and database design etc. | 7. Target is actual product-a unit, a module, a bent of integrated modules, and effective final product. |
| 8. Verification is done by QA team to ensure that the software is as per the specifications in the SRS document. | 8. Validation is carried out with the involvement of testing team. |
| 9. It generally comes first-done before validation. | 9. It generally follows after **verification**. |



[ ➔ examples of black and white box testing were done during the classes]

***** End of Chapter 6 *****

# Chapter: 7
# Metrics for Process and Products

## Software measurement

Measurement is the process by which numbers or symbols are assigned to attributes of entities in the real world in such a way as to describe them accordingly to clearly define unambiguous rules.

In physical world measure can be of two types:

1) **Direct Measure:**
   In direct measurement the product, process or thing is measured directly using standard scale. It includes cost and effort applied. Eg: lines of code produce, execution speed, memory size, defects reported/identified over certain period of time.
2) **Indirect Measure:**
   In indirect measurement the quantity or quality to be measured is measured using related parameter i.e. by use of reference. It includes functionality, quality, efficiency, complexity, reliability, maintainability etc.

## Need of Software Measurement

- To create the quality of the current product or process.
- To anticipate future qualities of the product or process.
- To enhance the quality of a product or process.
- To regulate the state of the project in relation to budget and schedule.

## Metrics for Software Quality

- Software metric are the numerical data related to software development.
- Software metric are quantifiable measure, that can be used to measure different characteristics of software system or the software development process.

### Categories of Metrics

1) Product Metrics:
   It describes the characteristics of the product such as size, complexity, design features, performance, efficiency etc. Product metrics are used to evaluate the state of the product, tracing risks and under covering prospective problem areas. The ability of team to control quality is evaluated.
2) Process Metrics:
   Process metrics pay particular attention on enhancing the long term process of the team or organization.





   It describes the effeteness and quality of the process that produces the software product. E.g.: effort required in process, time, number of defect found in testing, maturity of the process.
3) Project Metrics:
   It describes the project characteristics and its execution. E.g.: number of software, cost and schedule etc.

### Metrics for software project estimation

1) Line of code (LOC):
   - LOC as DSI (Delivered Source Instruction)
   - LOC as KDSI (Kilo Delivered Source Instruction)
2) Token Metrics:
   - Counting the token
   - Tokens can be if, for, switch etc.
3) Function point Metric (FP):
   - Size of software product is directly dependent on the number of different functions or features it supports.
   - Can be computed in three stages
     - ✓ UFP can be based on: number of inputs, outputs, inquires, number of files, external interfaces and their total count.
     - ✓ Compute the technical complexity factor (TCF).
       TCF can be based on performance Transaction Rate, reusability etc.
     - ✓ FP= UFP * TCF.
4) Feature Point Metric
   - Incorporates the algorithmic complexity of product.
5) Object metric, cohesion metrics, inheritance metrics etc.

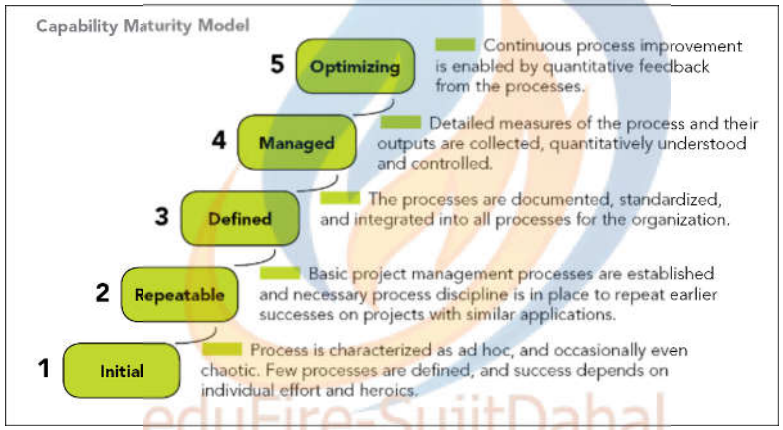
## Software Product Quality Standards => ISO 9000 Quality Model

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.

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.

1. **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.
2. **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.
3. **ISO 9003:** This standard applies to organizations that are involved only in the installation and testing of the products. For example, Gas companies.

## Software Process Quality Standards => Capability Maturity Model (CMM)/ Interface (CMMI):






**Level 1: Initial**

Ad hoc activities characterize a software development organization at this level. Very few or no processes are described and followed. Since software production processes are not limited, different engineers follow their process and as a result, development efforts become chaotic. Therefore, it is also called a chaotic level.





**Level 2: Repeatable**

At this level, the fundamental project management practices like tracking cost and schedule are established. Size and cost estimation methods, like function point analysis, COCOMO, etc. are used.

**Level 3: Defined**

At this level, the methods for both management and development activities are defined and documented. There is a common organization-wide understanding of operations, roles, and responsibilities. The ways through defined, the process and product qualities are not measured. ISO 9000 goals at achieving this level.

**Level 4: Managed**

At this level, the focus is on software metrics. Two kinds of metrics are composed.

**Product metrics** measure the features of the product being developed, such as its size, reliability, time complexity, understandability, etc.

**Process metrics** follow the effectiveness of the process being used, such as average defect correction time, productivity, the average number of defects found per hour inspection, the average number of failures detected during testing per LOC, etc. The software process and product quality are measured, and quantitative quality requirements for the product are met. Various tools like Pareto charts, fishbone diagrams, etc. are used to measure the product and process quality. The process metrics are used to analyze if a project performed satisfactorily. Thus, the outcome of process measurements is used to calculate project performance rather than improve the process.

**Level 5: Optimizing**

At this phase, process and product metrics are collected. Process and product measurement data are evaluated for continuous process improvement.

## Software Quality Assurance

- High quality software is an important goal.
- How do we define quality?

Many definitions of software quality have been proposed in the literature. Software quality is defined as: "Conformance to explicitly stated functions and performance requirements explicitly documented development standards and implicit characteristics that are expected to all professionally developed software".

Software Quality Assurance (SQA) is simply a way to assure quality in the software. It is the set of activities which ensure processes, procedures as well as standards suitable for the project and implemented correctly. Software Quality Assurance is a process which works parallel to development of a software. It focuses on improving the process of development of software so that problems can be prevented before they become a major issue. Software Quality Assurance is a kind of an Umbrella activity that is applied throughout the software process.

### SQA Activities:

Software quality assurance is composed of a variety of tasks 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:

**1. Prepares an SQA plan for a project:** The plan is developed during project planning and is reviewed by all stakeholders. Quality assurance activities performed by the software engineering team and the SQA group are governed by the plan. The plan identifies evaluations to be performed, audits and reviews to be performed, standards that are applicable to the project, procedures for error reporting and tracking, documents to be produced by the SQA team, and amount of feedback provided to the software project team.

**2. Participates in the development of the project's software process description:** The software team selects a process for the work to be performed. The SQA group reviews the process description for compliance with organizational policy, internal software standards, externally imposed standards (eg. ISO-9001), and other parts of software project plan.

**3. Reviews software engineering activities to verify compliance with the defined software process:** The SQA group identifies, documents, and tracks deviations from the process and verifies that corrections have been made.

**4. Audits designated software work products to verify compliance with those defined as a part of the software process :** The SQA group reviews selected work products, identifies, documents and tracks deviations, verifies that corrections have been made, and periodically reports the results of its work to the project manager.

**5. 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 plan, process description, applicable standards, or technical work products.

**6. Records any noncompliance and reports to senior management:** Non- compliance items are tracked until they are resolved.

## Software Reliability

Reliability is a measurable system attribute so non-functional reliability requirements may be specified quantitatively. These define the number of failures that are acceptable during normal use of the system or the time in which the system must be available.





**Functional reliability** requirements define system and software functions that avoid, detect or tolerate faults in the software and so ensure that these faults do not lead to system failure.

**Software reliability** requirements may also be included to cope with hardware failure or operator error.

- **Reliability:** The probability of failure-free system operation over a specified time in a given environment for a given purpose
- **Availability:** The probability that a system, at a point in time, will be operational and able to deliver the requested services

Software reliability emerged in the early 1970s and was created to predict the number of defects or faults in software as a method of measuring software quality. Software reliability is the 'probability that the software will execute for a particular period of time without failure, weighted by the cost to the user of each failure encountered'. Major types of reliability models include:

(a) finite versus infinite failure models

(b) static versus dynamic and

(c) deterministic versus probabilistic

Major types of dynamic reliability models include: life cycle versus reliability growth and failure rate, curve fitting, reliability growth,

For e.g. availability of 0.999 means that the system is up and running for 99.9% of the time.

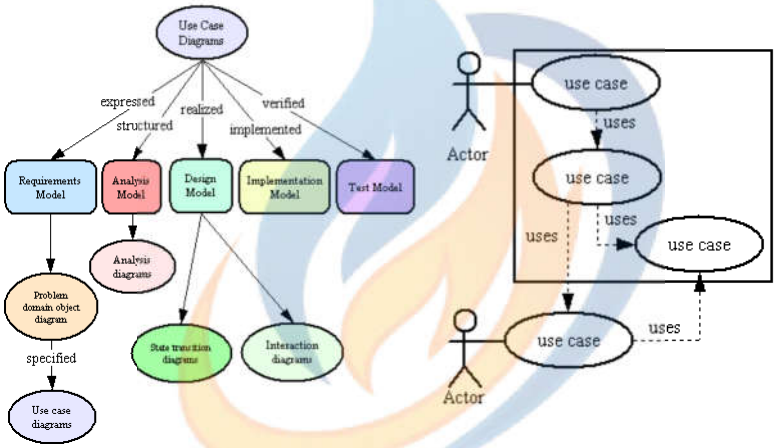
[ → some content were discussed during the classes]



***** End of Chapter 7 *****

# Chapter 8
# Introduction to Object Oriented Software Engineering (OOSE)

Object-Oriented Software Engineering (OOSE) is a software design technique that is used in software design in object-oriented programming.OOSE is developed by **Ivar Jacobson** in 1992. OOSE is the first object-oriented design methodology that employs use cases in software design. OOSE is one of the precursors of the Unified Modeling Language (UML), such as Booch and OMT. It includes a requirements, an analysis, a design, an implementation and a testing model. Interaction diagrams are similar to UML's sequence diagrams. State transition diagrams are like UML statechart diagrams.




UML stands for Unified Modeling Language. UML 2.0 helped extend the original UML specification to cover a wider portion of software development efforts including agile practices.

- Improved integration between structural models like class diagrams and behavior models like activity diagrams.
- Added the ability to define a hierarchy and decompose a software system into components and sub-components.
- The original UML specified nine diagrams; UML 2.x brings that number up to 13. The four new diagrams are called: communication diagram, composite structure diagram, interaction overview diagram, and timing diagram. It also renamed statechart diagrams to state machine diagrams, also known as state diagrams.





The current UML standards call for 13 different types of diagrams: class, activity, object, use case, sequence, package, state, component, communication, composite structure, interaction overview, timing, and deployment. These diagrams are organized into two distinct groups: structural diagrams and behavioral or interaction diagrams.

**Structural UML diagrams**

- Class diagram
- Package diagram
- Object diagram
- Component diagram
- Composite structure diagram
- Deployment diagram

**Behavioral UML diagrams**

- Activity diagram
- Sequence diagram
- Use case diagram
- State diagram
- Communication diagram
- Interaction overview diagram
- Timing diagram

## UML Diagram Symbols

There are many different types of UML diagrams and each has a slightly different symbol set.

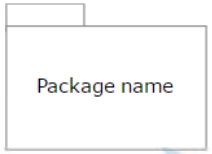
Class diagrams are perhaps one of the most common UML diagrams used and class diagram symbols center around defining attributes of a class. For example, there are symbols for active classes and interfaces. A class symbol can also be divided to show a class's operations, attributes, and responsibilities.

Visibility of any class members are marked by notations as on figure.

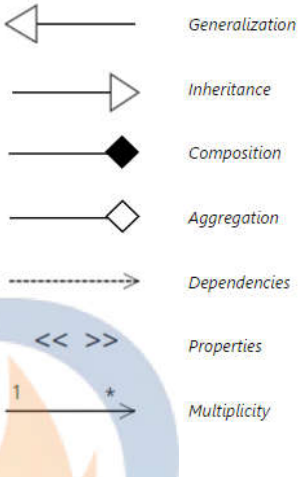
| | |
|---|---|
| + | *For Public* |
| - | *For Private* |
| # | *For Protected* |
| / | *For Derived* |
| ~ | *For Package* |

Lines are also important symbols to denote relationships between components. Generalization and Inheritance are denoted with empty arrowheads. Composition is shown with a filled in diamond. Aggregation is shown with an empty diamond. Dependencies are marked with a dashed line with an arrow. Using << >> allows you to indicate properties of that dependency. Multiplicity is usually shown with a number at one end of the arrow and a * at the other.

Package diagrams have symbols defining a package that look like a folder.




Activity diagrams have symbols for activities, states, including separate symbols for an initial state and a final state. The control flow is usually shown with an arrow and the object flow is shown with a dashed arrow.




Use case diagrams have symbols for actors and use cases.


**Why Do We Use UML?**

A complex enterprise application with many collaborators will require a solid foundation of planning and clear, concise communication among team members as the project progresses.

Visualizing user interactions, processes, and the structure of the system you're trying to build will help save time down the line and make sure everyone on the team is on the same page.

**Class Diagram**

Class diagrams are the backbone of almost every object-oriented method, including UML. They describe the static structure of a system.

**Use Case Diagram**

Use case diagrams model the functionality of a system using actors and use cases.

**Activity Diagram**

Activity diagrams illustrate the dynamic nature of a system by modeling the flow of control from activity to activity. An activity represents an operation on some class in the system that results in a change in the state of the system. Typically, activity diagrams are used to model workflow or business processes and internal operation.





**Sequence Diagram**

Sequence diagrams describe interactions among classes in terms of an exchange of messages over time.

**Interaction Overview Diagram**

Interaction overview diagrams are a combination of activity and sequence diagrams. They model a sequence of actions and let you deconstruct more complex interactions into manageable occurrences. You should use the same notation on interaction overview diagrams that you would see on an activity diagram.

**Timing Diagram**

A timing diagram is a type of behavioral or interaction UML diagram that focuses on processes that take place during a specific period of time. They're a special instance of a sequence diagram, except time is shown to increase from left to right instead of top down.

**Communication Diagram**

Communication diagrams model the interactions between objects in sequence. They describe both the static structure and the dynamic behavior of a system. In many ways, a communication diagram is a simplified version of a collaboration diagram introduced in UML 2.0.

**State Diagram**

Statechart diagrams, now known as state machine diagrams and state diagrams describe the dynamic behavior of a system in response to external stimuli. State diagrams are especially useful in modeling reactive objects whose states are triggered by specific events.

**Component Diagram**

Component diagrams describe the organization of physical software components, including source code, run-time (binary) code, and executables.

**Deployment Diagram**

Deployment diagrams depict the physical resources in a system, including nodes, components, and connections.

*** End of Chapter 8 ***