Dr. M.G.R. GOVERNMENT ARTS AND SCIENCE COLLEGE FOR WOMEN VILLUPURAM – 605 401.

SOFTWARE ENGINEERING - (BSCS55)

Objective:
This course introduces the concepts and methods required for the construction of large software intensive systems.

UNIT-I:
Introduction - Evolving Role of Software - Changing Nature of Software – Software Myths; A Generic View of Process: Layered Technology - Process Models: Waterfall Model - Evolutionary Process Models.

UNIT-II:
Requirements Engineering: Tasks - Initiating the Requirements Engineering Process - Eliciting Requirements - Building the Analysis Model - Requirements Analysis - Data Modelling Concepts.

UNIT-III:
Data Engineering: Design Process and Design Quality - Design Concepts - The Design Model Creating an Architectural Design: Software Architecture - Data Design -Architectural Design - Mapping Data Flow into Software Architecture; Performing User Interface Design: Golden Rules.

UNIT-IV:
Testing Strategies: Strategic Approach to Software Testing- Test Strategies for Conventional and Object Oriented Software - Validation Testing - System Testing -Art of Debugging. Testing Tactics: Fundamentals - White Box- Basis Path - Control Structure - Black Box Testing Methods

UNIT-V:
Project Management: Management Spectrum - People - Product - Process - Project. Estimation: Project Planning Process - Resources - Software Project Estimation - Project Scheduling - Quality Concepts - Software Quality Assurance - Formal Technical Reviews.

TEXT BOOK:

Roger S Pressman, ”Software Engineering - A Practitioner’s Approach”, Sixth Edition, McGraw Hill International Edition, New York: 2005.

REFERENCES:

1. Ian Somerville, “Software Engineering”, 7th Edition, Pearson Education, 2006.
2. Mall Rajib,” Software Engineering”, 2/E, PHI, 2006.

SOFTWARE ENGINEERING

UNIT -I

The changing nature of software

Software can be applied in a situation for which a predefined set of procedural steps. Based on complex growth of software it can be classified into following categories.

a) System software:

System software is a collection of programs used to run the system as an assistance to use other software programs. The compliers, editors, utilities operating system components, drivers and interfaces are example of system software.

b) Real-Time Software:

Real-Time software deals with changing environment. Fist it collect the input and convert it analog to digital, control component that responds to the external environment, perform the action in the last.
Ex: Air traffic control system, Flood control system, Network management system, etc.,

c) Embedded Software:

Software, when written to perform certain function under control conditions and further embedded into hardware as a part of large systems, is called embedded software.

d) Business software:

Software designed to process business applications is called business software, Business software could be a data and information processing application.

e) Personal Computer Software:

Word processing, spread sheet, computer graphics, multimedia, entertainment, database management, personal and business financial applications, external network or database access are only a few of hundreds of application.

f) Artificial Intelligence Software:

Artificial Intelligence software uses non-numerical algorithms, which use the data and information generated in the system, to solve the solving problem. Applications within this area include robotics, expert system, pattern recognition, artificial neural networks, theorem proving and game playing, signal processing software.

g) Web-based software:

Web-based software is the browsers by which web pages are processed i.e. HTML, Java, CGI, Perl, DHTML, etc.,

h) Engineering and scientific software:

Design, engineering of scientific software's deal with processing requirements in their specific fields. They are written for specific applications using the principals and formulas of each field.

Characteristics (main or prime objectives) of Software:

1. Maintainability
2. Correctness
3. Reusability
4. Reliability
5. Portability
6. Efficiency

1. Maintainability

Allows organizations to identify improvement areas as well as determine the value supplied by current applications or during development changes.

2. Correctness

The degree with which software adhears to its specified requirements.

3. Reusability

The case with which software can be reused in developing other software. The use of existing assets in some form within the software product development process.

4. Reliability

The portability of failure free software operations for a specified period of time in a specified environment.

5. Portability

Measures of how easily an application can be transferred from one computer environment to another.

6. Efficiency

Ability to avoid wasting materials, energy, efforts, money and time in during something or in producing a desired result.

Software Myth's?

There are some misbelieves in the software industry about the software and process of building software. For any software developer it is must to know beliefs and reality about them.

Here, we have the list of some common myths of software in software engineering according to the category:

1. Customer myths
2. Management myths
3. Developer Myths

1. Customer Myths

- The software myth believed by customer who can be internal or external.
- Customer believes that, general statement of objective is sufficient to begin writing programming.
- Change requirement is easy because software is flexible
- The customer always thinks that software development is an easy process.

2. Management myths

- Standard tools are present, they are sufficient for developers.
- They think, we can add more programs to easily they have latest computers.
- A good manager can manage any project.

3. Developer Myths

- They miss something. Now, I can fix it later.
- Once a program is written and running. My job is done.
- Until a program is running there is no way of accessing its quality.

Software Engineering – Layered Technology?

Software engineering is a fully a layered technology. To develop software, we need to go from one layer to another. All these layers are related to each other and each layer demands the fulfillment of the previous layer.

a. Quality focus:

The characteristics of good quality software are:

- Corrections of the function required to be performed by the software.
- Maintainability of the software
- Integrity i.e. providing security so that the unauthorized user cannot access information or data.
- Usability i.e. the efforts required to use or operate the software.

b. Process :

It is base layer or foundation layer for the software engineering:

- The software process is the key to keep all levels together.
- It defines a framework that includes different activities and tasks.
- In short, it covers all activities, actions and tasks required to be carried out for software development.

c.Methods:

- The method provides the answers of all 'how-to' that are asked during the process.
- It provides the technical way to implement the software.
- It includes collection of tasks starting from communication, requirement analysis, analysis and design modeling, program construction, testing and support.

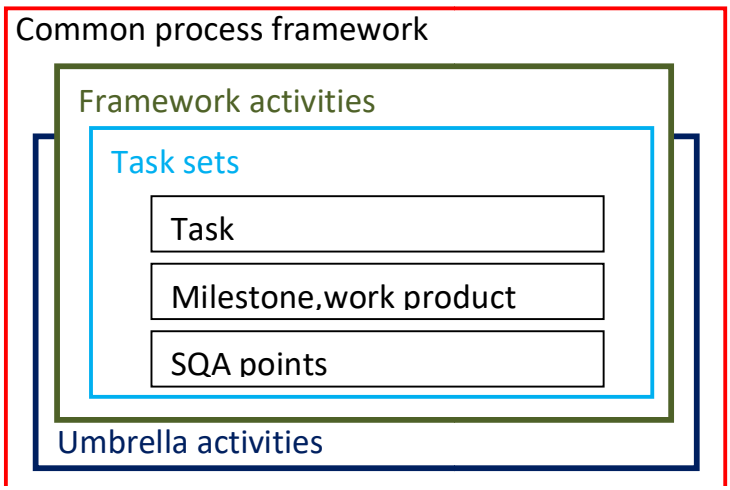
d. Tools:

- The software engineering tool is an automated support for the software development.
- The tools are integrated i.e. the information created by one tool can be used by the other tool.
- For example: The Microsoft publisher can be used as a verb designing tool.

Software Process Framework :

The process of framework defines a small set of activities that are applicable to all types of projects.

The software process framework is a collection of task sets. Task sets consist of a collection of small work tasks; project milestones, work productivity and software quality assurance points.



There are five generic process framework activities:

Communication

The software development starts with the communication between customer and developer.

Planning

It consists of complete estimation, scheduling for project development and tracking.

Modeling

Modeling consists of complete requirement analysis and the design of the project like algorithm, flowchart etc. The algorithm is the step-by-step solution of the problem and the flow chart shows a complete flow diagram of a program:

Construction

- Construction consists of code generation and the testing part.
- Coding part implements the design details using an appropriate programming language.
- Testing is to check whether the flow of coding is correct or not.
- Testing also check that the program provides desired output.

Deployment:

- Deployment step consists of delivering the product to the customer and take feedback from them.
- If the customer wants some corrections or demands for, the additional capabilities, then the change is required for improvement in the quality of the software.

The Umbrella activities in software engineering process:

Typical umbrella activities are:

1. Software project tracking and control
   - In this activity, the developing team, accesses project plan and compares it with the predefined schedule.
   - If these project plans do' not match with the predefined schedule, then the required actions are taken to maintain the schedule.

2. Risk management
   - Risk is an event that may or may not occur.
   - If the event occurs, then it causes some unwanted outcome. Hence, proper risk management is required.

3. Software Quality Assurance (SQA)
   - SQA is the planned and systematic pattern of activities which are required to give a guarantee of software quality.

For example, during the software development meetings are conducted at every stage of development to find out the defects and suggest improvements to produce good quality software.

4. Formal Technical Reviews (FTR)
   - FTR is a meeting conducted by the technical staff.
   - The motive of the meeting is to detect quality problems and suggest improvements.
   - The technical person focuses on the quality of the software from the customer point of view.

5. Measurement
   - Measurement consists of the effort required to measure the software.
   - The software cannot be measured directly. It is measured by direct and indirect measures.
   - Direct measures like cost, lines of code, size of software etc.
   - Indirect measures such as quality of software which is measured by some other factor.

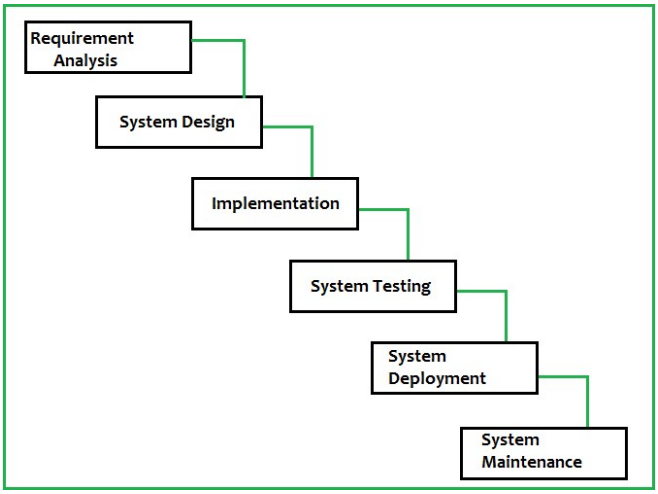
6. Software Configuration Management (SCM)
   - It manages the effect of change throughout the software process.

7. Reusability management
   - It defines the criteria for reuse the product. The quality of software is good when the components of the software are developed for certain application and are useful for developing other applications.

8. Work product preparation and production
   - It consists of the activities that are needed to create the documents, forms, lists, logs and user manuals for developing software.

"The Waterfall Model":

The waterfall model is also called as "Linear Sequential Model" or "Classical life cycle model". In this model, each phase is fully completed before the beginning of the next phase. This model is used for the small projects. In this model, feedback is taken after each phase to ensure that the project is on the right path.



Requirement analysis and specification:

- The aim of the requirement analysis and specification phase is to understand the exact requirement of the customer and document them properly.
- Firstly all the requirements regarding the software are gathered from the customer and then the gathered requirements are analyzed.
- These analyzed requirements are documented in a software requirement specification (SRS) document. SRS document serves as a contract between development team and customers.

Design:

The aim of the design phase is to transform the requirements specified in the SRS document into a structure that is suitable for implementation in some programming language.

Coding and testing:

In coding phase software design is translated into source code using any suitable programming language. Thus each designed module is coded. The aim of the testing phase is to check whether each module is working properly or not.

Deployment :

Deploy the application in the respective environment.

Maintenance:

There are some issues which come up in the client environment. To fix those issues, patches are released. Also to enhance the product some better versions are released. Maintenance is one to deliver these changes in the customer environment.

Advantages:

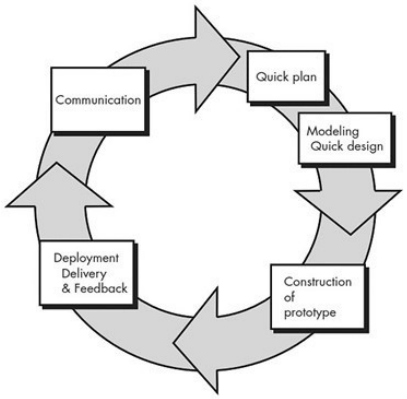
- Before the nest phase of development, each phase must be completed.
- Suited for smaller projects when requirements are well defined.
- They should perform quality assurance test (validation and verification) before completing each stage.
- Elaborate documentation is done at every phase of the software development cycle.
- Project is completely dependent on project team with minimum client intervention.
- Any changes in software are made during the process of the development.

Disadvantages:

- Error can be fixed only during the testing phase.
- It is not desirable for complex project where requirement changes frequently.
- Test period comes quite late in the development process.
- Documentation occupies a lot of time of developers and testers.
- Client valuable feedback cannot be included with ongoing development phase.
- Small changes or errors that arise in the completed software may cause a lot of problems.

" Prototype Model":

The Prototyping Model is a Systems Development Methodology (SDM) within which a paradigm output (or an early approximation of a final system or product) is constructed, tested, and then reworked. It is done till an appropriate paradigm is achieved to help develop the entire system or product. This model works best in situations when all the details or requirements are not known well in advance. It is majorly a trial-and-error process which works in an iterative fashion.



The different phase of prototype model are:

a. Communication
b. Quick design
c. Modeling quick design
d. Construction of prototype
e. Deployment, delivery, feedback

a. Communication:

In this phase, developer and customer meet and discuss the overall objectives of the software.

b. Quick Design:

Quick design is implemented when requirement are known. It includes only the important aspects i.e. input and output format of the software. It focuses on those aspects which are visible to the user rather than the detailed plan. It helps to construct a prototype.

c. Modeling quick design:

This phase gives the clear idea about the development of software as the software is now constructed. It allows the developer to better understand the exact requirements.

d. Construction of prototype:

The prototype is evaluated by the customer itself.

e. Deployment, delivery, feedback:

If the user is not satisfied with current prototype then it is refined according to the requirements of the requirements of the user. The process of refining the prototype is repeated till all the requirements of users are met. When the users are satisfied with the developed prototype than the system developed on the basis of final prototype.

Advantages:

- Prototype model need not know the detailed input, output, process, adaptability of operating system and full machine iteration.
- In the development process of this model users are actively involved.
- The development process is the best platform to understand the system by the user.
- Errors are detected much earlier.
- Gives quick user feedback for better solutions.
- It identifies the missing functionality easily. It also identifies the confusing or difficult functions.

Disadvantages:

- The client involvement is more and it is not always considered by the developer.
- It is a slow process because it takes more time for development.
- Many changes can disturb the rhythm of the development team.
- It is a throwaway prototype when the users are confused with it.

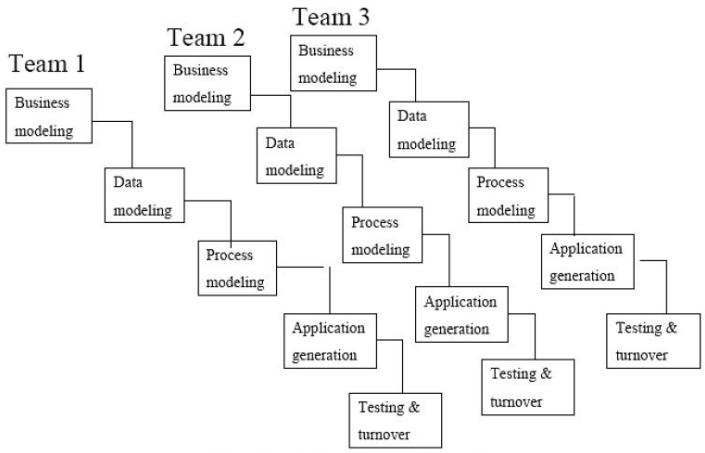
"RAD Model":

RAD is a Rapid Application Development Process model. Using the RAD mode, software product is developed in a short period of time. The initial activity starts with the communication between customer and developer.

Planning depends upon the initial requirements and then the requirements are divided into groups. Planning is more important to work together on different modules.

The RAD model consists of the following phases:

a) Business Modeling
b) Data modeling
c) Process Modeling
d) Application generation
e) Testing and turnover



a) Business Modeling:

Business modeling consists of the flow of information between various functions in the project. For example, what type of information is produces by every function and which are the functions to handle that information. It is necessary to perform complete business analysis to get the essential business information.

b) Data modeling:

The information in the business modeling phase is refined into the set of objects and it is essential for the business. The attributes of each object are identified and defined the relationship between objects.

c) Process modeling:

The data objects defined in the data modeling phase are changed to fulfill the information flow to important the business model, they process description is created for adding, modifying, deleting or retrieving a data object.

d) Application generation:

In the application generation phase, the actual system is built. To construct the software the automated tools are used.

e) Testing and turnover:

The prototypes are independently tested after each iteration so that the overall testing time is reduced. The data flow and the interfaces between all the components are fully tested.

Advantages:

- The Processes of application development and delivery are fast.
- The model is flexible, if any changes are required.
- Reviews are taken from the clients at the starting of the development hence there are lesser chances to miss the requirements.

Disadvantages:

- The feedback from the user is required every development phase.
- This model is not a good choice for long term and large projects.

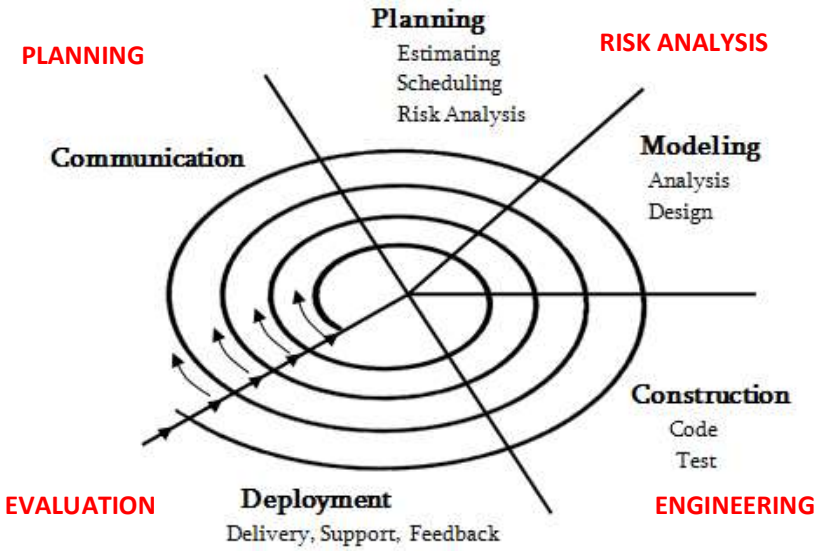
"Spiral Model":

The spiral model is similar to the incremental development for a system, with more emphasis placed on risk analysis. The spiral model has four phases: planning, design, construction and evaluation. A software project repeatedly passes through these phases in iteration.

It is a combination of prototype and sequential or waterfall model. This model was developed by Boehm. It is used for generating the software projects. This model is risk driven process model.

Every phase in the spiral model is start with a design goal and ends with the client review. The development team in this model begins with a small set of requirements and for the set of requirements team

goes through each development phase. The development team adds the functionality in every spiral till the application is ready.



Following are the steps involved in spiral model:

a) Planning
b) Risk analysis
c) Engineering
d) Evaluation

a) Planning:

This phase studies and collects the requirements for continuous communication between the customer and system analysis. It involves estimating the cost and resources for the iteration.

b) Risk analysis:

This phase, identifies the risk and provides the alternative solutions if the risk is found.

c) Engineering:

In this phase, actual development i.e. coding of the software is completed. Testing is completed at the end of the phase.

d) Evaluation:

Get the software evaluated by the customers. They provided the feedback before the project continues to the next spiral.

Advantages:

- It reduces high amount of risk.
- It is good for large and critical projects.
- It gives strong approval and documentation control.

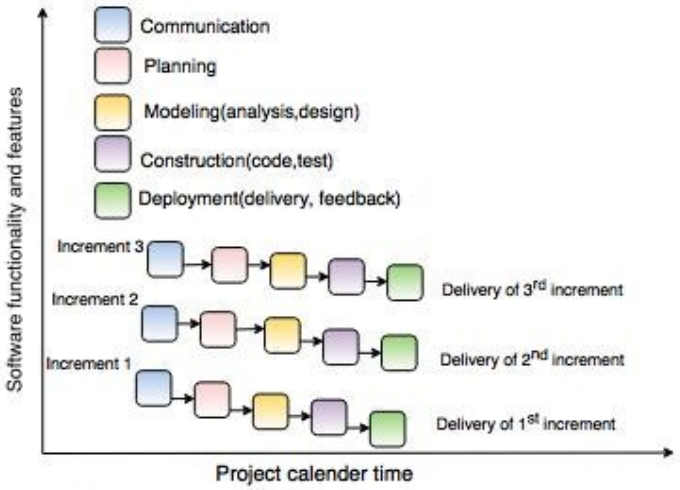
Disadvantages:

- It can be costly to develop a software model.
- It is not used for small projects.

"Incremental Model":

The incremental model combines the elements of waterfall model and they are applied in an iterative fashion. The first increment in this model is generally a core product.

Each increment builds the product and submits it to the customer for suggesting any modifications. The next increments implement the customer's suggestions and add additional requirements in the previous increment. This process is repeated until the product is completed.



Communication:

The software development starts with the communication between customer and developer.

Planning:

It consists of complete estimation, scheduling for project development.

Modeling:

Modeling consists of complete requirements analysis and the design of the project like algorithm, flow chart etc., and the algorithm is a step-by-step solution of the problem and the flow chart shows a complete flow diagram of a program.

Construction:

Construction consists of code generation and the testing part. Coding part implements the design details using an appropriate programming language.

Testing is to check whether the flow of coding is correct or not, Testing also checks that the program provides desired output.

Deployment:

Deployment step consists of delivering the product to the customer and taking feedback from them. If the customer wants some corrections or demands for the additional capabilities, than the change is required for improvement in the quality of the software.

Advantages:

- This model is flexible because the initial product delivery is faster.
- It is easier to test and debug in the smallest iteration.
- The working software is generated quickly in the software life cycle.
- The customers can respond to its functionalities after every increment.

Disadvantages:

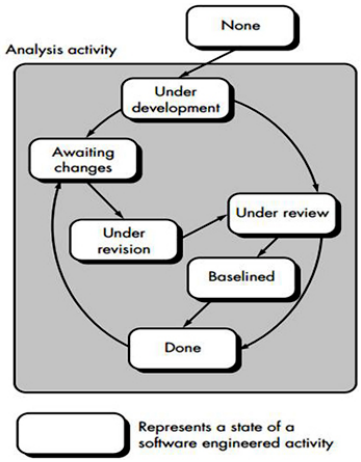
- The cost of the final product may cross the cost initially estimate.
- The planning of design is required before the whole system is broken into smaller increments.

The demands of customer for the additional functionalities after every increment causes problem in the system architecture.

"Concurrent Development Process Model":

Concurrent process model is an evolutionary process model in software engineering. The term concurrent means "done at the same time". If we take waterfall model as an example, you will not know the activities going on in each phase, only the phase is over, you get a work product or document.

Suppose we want to know the state of activities of the design phase and at the same time we want to know about testing phase activities. Which once are complete and which are not? The concurrent process model shows the current state of activities, tasks and their associated states that remain in different phases.



'Design phase' may be at an 'awaiting state' and 'customer communication' is in 'under revision' state. The customer wants some changes to the design, than 'communication' goes to 'awaiting changes' and design goes to the 'under revision' stage again. The benefit of this model is that project managers know each phase is what state and why.

Question and Answer - 2 – Mark

1. Define "Software Engineering" ?

The term Software Engineering is composed of two words, 'Software Engineering'. Software is the set of instructions or programme instructing a computer to do specific task. Software is considered to be a collection of executable programming code, associated libraries and documentation.

Software Engineering is an engineering branch associated with the development of software product using well-defined scientific principals, methods and procedures.

2. What are the characteristics of Software Engineering?

Software development is a logical activity and therefore it is important to understand basic characteristics of software.

- Software is engineered (developed), not manufactured.
- Software does not wear out.
- Software is highly flexible.
- Most software is created & not assembled from existing components.

3. Difference between process and methods

Process :

It is base layer or foundation layer for the software engineering:

- The software process is the key to keep all levels together.
- It defines a framework that includes different activities and tasks.
- In short, it covers all activities, actions and tasks required to be carried out for software development.

Methods:

- The method provides the answers of all 'how-to' that are asked during the process.
- It provides the technical way to implement the software.
- It includes collection of tasks starting from communication, requirement analysis, analysis and design modeling, program construction, testing and support.

4. List the process models for software development?

- The Waterfall model
- Prototyping model
- Rapid Application Development model
- Evolutionary process model
  - Spiral model
  - Increment model
  - Concurrent Development model

5. Define System Software?

System software is the collection of programs used to run the system as an assistance to use other software programs. The compilers, editors, utilities operating system components, drivers a d interfaces are example of the system software.

6. Define software myth?

There are some misbelieves in the software industry about the software and process of building software. For any software developer it is must to know such beliefs and reality about them. There are three types of myth:

1. Management Myth
2. Customer Myth
3. Developer or Practitioner Myth

7. What are the processes of a software development?

The process of a software development has three Generic views which are:

1. Definition Phase
2. Development Phase
3. Maintenance Phase

8. Define Software Process?

Software process can be defined as the structured set of activities that are required to develop the software system.

9. Define Waterfall Model?

The waterfall model is also called as "Linear Sequential Model" or "Classical life cycle model". In this model, each phase is fully completed before the beginning of the next phase. This model is used for the small projects. In this model, feedback is taken after each phase to ensure that the project is on the right path.

10. Define prototype?

Prototype is defined as first or preliminary from using which other forms are copied or derived. Prototype model is a set of general objectives for software.

11. What are the different phases of prototyping model?

The different phases of prototyping model are:

1. Communication
2. Quick design
3. Modeling and quick design
4. Construction of prototype
5. Deployment, delivery, feedback.

12. Define RAD?

RAD is a Rapid Application Development Process model. Using the RAD mode, software product is developed in a short period of time. The initial activity starts with the communication between customer and developer.

13. Define Spiral Model?

The spiral model is similar to the incremental development for a system, with more emphasis placed on risk analysis. The spiral model has four phases: planning, design, construction and evaluation. A software project repeatedly passes through these phases in iteration.

14. Define incremental model?

The incremental model combines the elements of waterfall model and they are applied in an iterative fashion. The first increment in this model is generally a core product.

15. Define concurrent development process model?

Concurrent process model is an evolutionary process model in software engineering. The term concurrent means “done at the same time”.

UNIT – II

Requirement engineering process:

a) Feasibility study
b) Requirement Elicitation and Analysis
c) Software Requirement Specification
d) Software Requirement Validation
e) Software Requirement Management

a) Feasibility study

The objective behind the feasibility study is to create the reason for developing the software that is acceptable to users, flexible to change and comfortable to established standards.

Types of Feasibility

a) **Technical Feasibility**
   Technical feasibility evaluates the current technologies, which are needed to accomplish customer requirements with the time and budget.
b) **Operational Feasibility**
   Operational feasibility assesses the range in which the required software performs a series of level to solve business problem and customer requirements.
c) **Economic feasibility**
   Economic feasibility decides whether the necessary software can generate financial profits for an organization.

b) Requirement Elicitation and Analysis

Analysis of requirements starts with requirement elicitation. The requirements are analyzed to identify inconsistence, defects, omission, etc. We describe requirements in terms of relationships and also resolve conflicts if any.

Problems of elicitation and analysis:

- Getting all, and only, the right people involved.
- Stakeholders often don't know what they want
- Stakeholders express requirements in their terms.
- Stakeholders may have conflicting requirements.
- Requirements changes during the analysis process.
- Organizational and political factors may influence system requirements.

c) Software Requirement Specification:

Software requirement specification is a kind of document which is created by a software analyst after the requirements collected from the various sources the requirement received by the customer written in ordinary language. It is the job of the analyst to write the requirement in technical language so that they can be understand and beneficial by the development team.

The models used at this stage include ER diagram, data flow diagrams (DFDs), function decomposition diagrams (FDDs), data dictionaries, etc.

Data Flow Diagrams:

Data flow Diagrams (DFDs) are used widely for modeling the requirements. DFD shows the flow of data through a system. The system may be a company, an organization, a set of procedures, a

computer hardware system, a software system, or any combination of the preceding. The DFD is also known as a data flow graph or bubble chart.

Data dictionaries:

Data dictionaries are simply repositories to store information about all data items defined in DFDs. At the requirements sage, the data dictionary should at least define customer data items, to ensure that the customer and developers use the same definition and technologies.

Entity-Relationship (ER) Diagrams:

Another tool for requirement specification is the entity-relationship diagram, often called an "E-R diagram". It is detailed logical representation of the data for the organization, and three main constructs i.e. data entities, relationships and their associated attributes.

d) Software Requirement Validation:

After requirement specification developed, the requirement discussed in this document are validated. The user might demand illegal, impossible solution or experts may misinterpret the needs. Requirements can be check against the following conditions:

- ✓ If they can practically implement
- ✓ If they are correct and as per the functionality and specially of software.
- ✓ It there are any ambiguities
- ✓ If they are full
- ✓ If they can describe

Requirement validation techniques

- **Requirements reviews/ inspections**: systematic manual analysis of the requirement.
- **Prototyping :** using an executable model of the system to check requirements.
- **Test-case generation:** Developing tests for requirements to check testability.
- **Automated consistency analysis :** checking for the consistency of structured requirements descriptions.

e) Software Requirement Management

Requirement management is the process of managing changing requirements during the requirements engineering process and system development. New requirement emerge during the process as business needs a change, and a better understanding of the system is developed. The Priority of requirements from different viewpoints changes during development process. The business and technical environment of the system changes during the development.

Software Requirements:

Largely software requirements must be categorized into two categories:

a. **Financial Requirements**: Financial requirements define a function that a system or system element must be qualified to perform and must be documented in different forms. The functional requirements are describing the behavior of the system as it correlates to the system's functionality.
b. **Non-functional Requirements**: This can be the necessities that specify the criteria that can be used to decide the operation instead of specific behaviors of the system.
   - Execution qualities like security and usability, which are observable at run time.
   - Evolution qualities like testability, maintainability, extensibility, and scalability that embodied in the static structure of the software system.

Requirement Engineering Tasks:

Requirement engineering consists of seven different tasks as follow:

1. Inception:

Inception is a task where the requirement engineering asks a set of question to establish a software process. In this task, it understands the problem and evaluates with the proper solution.

It collaborates with the relationship between the customer and the developer. The developer and customer decide the overall scope and the nature of the question.

2. Elicitation:

Elicitation means to find the requirements from anybody.

The requirements are different because the following problem occur in elicitation.

Problem of scope: The customer gives the unnecessary techniques detail rather than clarity of the overall system objective.

Problem of understanding : poor understanding between the customer and the developer regarding various aspect of the project like capability, limitation of the computing environment.

Problem of volatility: In this problem, the requirements change from time to time and it is difficult while developing the project.

3. Elaboration:

In this task, the information taken from user inception and elaboration and are expanded and refined in elaboration. Its main task is developing pure model of software using functions, feature and constraints of a software.

4. Negotiation:

In Negotiation task, a software engineer decides the how will the project be achieved with limited business resources. To create rough gusses of development and access the impact of the requirement on the project cost and delivery time.

5. Specification:

In this task, the requirement engineer constructs a final work product. The work product is in the form of software requirement specification. In this task, formalize the requirement of the proposed software such as informative, functional and behavioral. The requirement are formalize in both graphical and textual formats.

6. Validation:

The work product is built as an output of the requirement engineering and that is assessed for the quality through a validation step. The formal technical reviews from the software engineer, customer and other stakeholder helps for the primary requirements validation mechanism.

7. Requirement Management:

It is a set of activities that help the project team to identify, control and track the requirements and changes can be made to the requirements at any time of the ongoing project. These tasks stats with the identification and assign a unique identifier to each of the requirement.

After finalizing the requirement traceability table is developed, The examples of traceability table are the feature, sources, dependencies, subsystem and interface of the requirement.

The requirements engineering process:

The process of initiating engineering is the subject of this section. The point of view describe is having all stakeholders (including customers and developers) involved in the creation of the system requirements documents. The following are steps required to initiate requirements engineering.

Identifying the stakeholders:

- A stakeholder is anyone who benefits in a direct or indirect way from the system which is being developed.
- In the book stakeholders are: operations managers, product managers, marketing people, internal and external customers, end-users, consultants, product engineers, software engineers, support and maintenance engineers etc.
- At inception, the requirements engineer should create a list of people who will contribute input as requirements are elicited.

Recognizing multiple viewpoints:

- Each of the stakeholders will contribute to the Requirement engineering process.
- As information from multiple viewpoints is collected, emerging requirements may be inconsistent or may conflict with one another.
- The requirements engineer is to categorize all stakeholder information including inconsistencies and conflicting requirements ina way that will allow decision makers to choose an internally inconsistent set of requirements for the system.

Working towards collaboration:

Collaboration does not necessarily mean that requirements are defined by committee. In many cases, stakeholders collaborate by proving their view of requirements, but a strong " project champion" (business manager, or a senior technologist) may make the final decision about which requirements make the final cut.

Eliciting Requirements?

Eliciting requirements helps the user for collecting the requirements.

a) Collaborative requirements gathering:

Gathering the requirements by conducting the meeting between developer and customer. Fix the rules for preparation and participation.

The main motive is to identify the problem, give the solutions for the elements, negotiate the different approaches and specify the primary set of solution requirements in an environment which is valuable for achieving goal.

b) Quality Function Deployment (QFD):

In this technique, translate the customer need into the technical requirement for the software. QFD system designs software according to the demands of the customer.

QFD consist of three types of requirement:

i. Normal requirements:

Te objective and goal are stated for the system through the meetings with the customer. For the customer satisfaction these requirements should be there.

ii. Expected requirement:

These requirements are implicit. These are the basic requirement that nor be clearly told by the customer, but also the customer expect the requirement.

iii. Exciting requirement:

These features are beyond the expectations of the customer. The developer adds some additional features or unexpected feature into the software to make the customer more satisfied. For example, the mobile phone with standard features, but the developer adds little additional functionality like voice searching multi-touch screen etc. than the customer more excited about the feature.

c) Usage scenarios:

Till the software team does not understand how the features are function are used by the end users it is difficult to move technical activities, To achieve above problem the software team products a set of structure that identity the usage for the software, This structure is called as 'Use cases'.

d) Elicitation work product:

The work product created as a result of requirement elicitation that is depending on the size of the system or product to be built. The work product consists of a statement need, feasibility and statement scope for the system. It is also consists of a list of users participate in the requirement elicitation. Analysis model operates as like between the 'system description' and the 'design model'.

In the analysis model, information, functions and the behavior of the system is defined and these are translated into the architecture, interface and component level design in the 'design modeling'.

Building the analysis model? (or) Discuss in detail the basic structure of analysis model?

The instant of the analysis model is to provide a description of the required information, functional and behavioral domains for a computer-based system. The model changes dramatically as software engineers learn more about the system, and the stakeholders understand more about what they really require.

a) Scenario-based elements:

The system is described from the user's point of view using a scenario-based approach. It always a good idea to get stakeholders involved. One of the best ways to do this is to have get stakeholders write use-cases that describe how the software engineering models will be used. Functional- processing narratives for software functions. Use-case descriptions of the interaction between an actor and the system.

b) Class-based elements:

Each usages scenario implies a set of objects that are manipulated as an actor interacts with the system. These objects are categorized into classes a collection of things that have similar attributes and common behavior. One way to isolate classes is to look for descriptive nouns in a use-case script. At least some of the nouns will be candidate classes.

c) Behavioral elements:

This state diagram is one method for representing the behavior of a system by depicting its states and the events that cause the system to change state. A state is any observable mode of behavior. Moreover, the state diagram indicated what actions are taken as a consequence of a particular event. Flow-oriented elements: information is transformed as it flows through a computer-based system. The system accepts input in a variety of forms; applies functions to transform it; and procedures output in a variety of forms.

The concepts of data modeling?

Data modeling is the basic step in the analysis modeling. In data modeling the data objects are examined independently of processing. Data modeling sometimes also called information modeling. It is the process of visually representing what data the application or the system will use and how it will flow.

The data domain is focused and a model is created at the customer level of abstraction. The data model represents how data objects are related with one another. The fundamental element that a data model needs to includes and describe about the data objects, entities, attributes and the relationships.

a) Data Object:

Data object is a set of attributes that will be manipulated with the software system. Each instance of data instance of data object can be identified with the help of unique identifier. For example, A student can be identified by using is roll number.

Object : Vehicle
Attributes:
Make:
Model:
Color:
Owner:
Price:

The system cannot perform without accessing to the instances of object. Each data object is described by the attributes which themselves are data items. From the diagram the vehicle is data is data object which can be defined or viewed with the help of set attributes.

b) Attributes:

Attributes defines properties of data object. Typically there are three types of attributes.

- Naming attribute
- Descriptive attribute
- Referential attribute

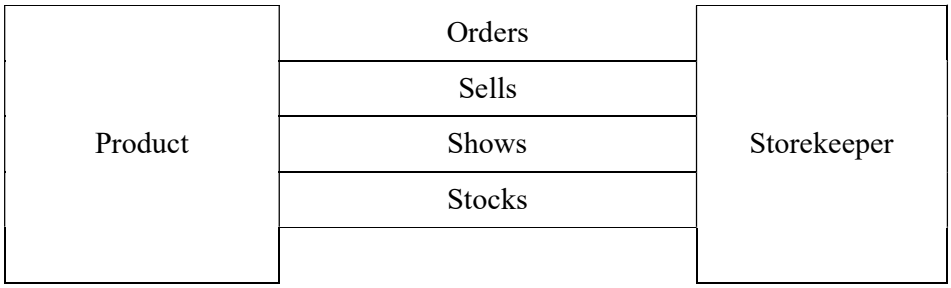
Naming attribute: These attribute are used to name an instance of data object. For example, in a vehicle data object make and model are naming attribute.

Descriptive attribute: These attributes are used to describe the characteristics or feature of the data object. For example, in a vehicle data object color is a descriptive attribute.

Referential attribute: These are the attributes that ate used in making te reference to another instance in another table. For example, in a vehicle data object owner is a referential attribute.

c) Relationship:

Relationship represents the connection between the data objects. For example, the relationship between a product and a storekeeper is show table



Hence the product and the storekeeper are two objects that share following relationships.

- Storekeeper orders product.
- Storekeeper sells product
- Storekeeper shows product
- Storekeeper stocks product

Question and Answer - 2 – Mark

1. Define Requirement Engineering?

- Requirement engineering refers to the process of defining, documenting, and maintaining requirements in the engineering design process.
- Requirement engineering provides the appropriate mechanism to understand what the customer desire.
- The process of collecting the software requirement from the client than understand, evaluate and document it is called as requirement engineering.
- Requirement engineering constructs a bridge for design and construction.

2. What is feasibility study?

The objective behind the feasibility study is to create the reason for developing the software that is acceptable to users, flexible to change and comfortable to established standards.

3. What the types of feasibility?

d) **Technical Feasibility**

Technical feasibility evaluates the current technologies, which are needed to accomplish customer requirements with the time and budget.

e) **Operational Feasibility**

Operational feasibility assesses the range in which the required software performs a series of level to solve business problem and customer requirements.

f) **Economic feasibility**

Economic feasibility decides whether the necessary software can generate financial profits for an organization.

5. Define Data dictionaries?

Data dictionaries are simply repositories to store information about all data items defined in DFDs. At the requirements sage, the data dictionary should at least define customer data items, to ensure that the customer and developers use the same definition and technologies.

6. Define Entity-Relationship Diagrams?

Another tool for requirement specification is the entity-relationship diagram, often called an "E-R diagram". It is detailed logical representation of the data for the organization, and three main constructs i.e. data entities, relationships and their associated attributes.

7. Define Software Requirement Management?

Requirement management is the process of managing changing requirements during the requirements engineering process and system development.

8. Define Quality Function Deployment?

In this technique, translate the customer need into the technical requirement for the software. QFD system designs software according to the demands of the customer.

9. What the non-functional requirements of software?

Non-functional Requirements can be the necessities that specify the criteria that can be used to decide the operation instead of specific behaviors of the system.

- Execution qualities like security and usability, which are observable at run time.
- Evolution qualities like testability, maintainability, extensibility, and scalability that embodied in the static structure of the software system.

10. Define Requirement analysis?

Requirement analysis helps to understand, interpret, classify, and organize the software requirements in order to assess the feasibility, completeness and consistency of the requirements.

11. Define Data Modeling?

Data modeling is the basic step in the analysis modeling. In data modeling the data objects are examined independently of processing. Data modeling sometimes also called information modeling. It is the process of visually representing what data the application or the system will use and how it will flow.

12. Define Data Object?

Data object is a set of attributes that will be manipulated with the software system. Each instance of data instance of data object can be identified with the help of unique identifier. For example, A student can be identified by using is roll number.

13. What is data dictionary?

Data dictionaries are simply repositories to store information about all data items defined in DFDs. At the requirements sage, the data dictionary should at least define customer data items, to ensure that the customer and developers use the same definition and technologies.

14. Define Attributes and its types ?

Attributes defines properties of data object. Typically there are three types of attributes.

- Naming attribute
- Descriptive attribute
- Referential attribute

15. Define Data flow Diagram?

Data flow diagram is graphical representation of flow of data in an information system. It is capable of depicting incoming data flow, outgoing data flow and stored data. The DFD does not mention anything about how data flows though the system.

16. What are the types of DFD?

- **Logical DFD:** This type of DFD concentrates on the system process, and flow of the data in the system. For Example in a banking software system, how data is moved between different entities.
- **Physical DFD:** This type of DFD shows how the data flow is actually implemented in the system. It is more specific and close to the implementation.

17. What are the components of DFD?

DFD can represent sources, destination, storage and flow of the data using the following set of components:

- **Entities:** Entities are source and destination of information date. Entities are represented by a rectangle with their respective names.
- **Process:** Activities and action taken on the data are represented by cicle or round edged rectangles.
- **Data Storage:** There are two variants of data storage. It can either be represented as a rectangle with absence of both smaller sides or as an open-side rectangle with only one side missing.
- **Data Flow:** Movement of data is shown by pointed arrows Data movement is shown from the base of arrow as its sources towards head of the arrow as destination.

18. Write Short notes on data modeling?

Data modeling is the basic step in the analysis modeling. In data modeling the data objects are examined independently of processing. Data modeling sometimes also called information modeling. It is the process of visually representing what data the application or the system will use and how it will flow.

The data domain is focused and a model is created at the customer level of abstraction. The data model represents how data objects are related with one another. The fundamental element that a data model needs to includes and describe about the data objects, entities, attributes and the relationships.

UNIT – III

Data Engineering

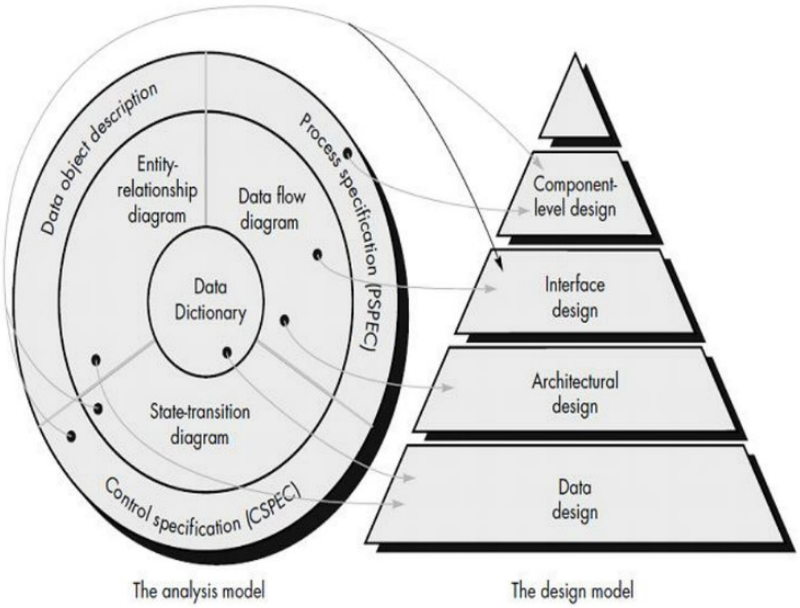
The intent of this chapter is to provide an introduction to the design process and to describe fundamental design concepts that are essential to an understanding of any software design method.

The design model consists of the data design, architectural design, interface design, and component-level design. The goal of design engineering is to produce a model or representation that exhibits firmness, commodity, and delight.

Design within the contest of software engineering:

Software design is the last software engineering action within the modeling activity and sets the stage for construction such as code generation and testing.

The analysis mode, manifested by scenario-based, class-based, flow-based and behavioral elements, feed the design task.



Data Design:

The data design defines to transform the information domain model of analysis phase into the data structure. These structures play an important role in software implementation. The E-R Diagram and data dictionary are used to create the data design model.

Architectural Design:

The architectural design defines the relationships between more structural elements of the software, the architectural style and design patterns that can be used to achieve the requirements defined for the system, and the constrains that affect the way in which the architectural design can be implemented. This design can be derived from the system specs, the analysis model, and interaction of subsystems defined within the analysis model.

Interface Design:

The interface design describes how the software communicates with system that interpolate with it, and with humans who use it. An interface implies a flow of information (data and control) and a specific type of behavior.

Component-Level Design:

The component –level design transforms structural elements of the software architecture into a procedural description of software components.

Design Process and design Quality

Software design is an iterative (Step by step) process though which requirements are translated into a 'blueprint' for constructing the software. Initially, the blueprint depicts a holistic view of software, i.e. the design is represented at a high-level of abstraction.

Three characteristics serve as a guide for the evaluation of a good design:

1. The design must implement all of the explicit requirements contained in the analysis mode, and it must accommodate all of the implicit requirements desired by the customer.
2. The design must be a readable, understandable guide for those who generate code and for those who test subsequently support the software.
3. The design should provide a complete picture of the software, addressing the data, functional, and behavioral domains from an implementation perspective.

Quality Guidelines

In order to evaluate the quality of a design representation, we must establish technical criteria for good design.

- A design should exhibit architecture that: has been created using recognizable architectural styles or patterns. Is composed of components that exhibit good design characteristics.
- A design should be modular; that is the software should be logically partitioned into elements or subsystem.
- A design should contain distinct representation of data, architecture, interface and components.
- A design should lead to data structures that are appropriate for the classes 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 connection between components and with the external environment.
- A design should be derived using a repeatable method that is driven by information obtained during software requirements analysis.
- A design should be represented using a notation that effectively communications its meaning.

Quality Attributes:

The design quality attributes that has been given the acronym **FURPS**. The FURPS quality attributes represent a target for all software design.

- **Functionality:** is assessed by evaluating the features set and capabilities of the program, the generality of the functions that are delivered, and the security of the overall system.
- **Usability:** is assessed by considering human factors, overall aesthetics, constancy and documentation.
- **Reliability:** is evaluated by measuring the frequency and severity of failure, the accuracy of output results, the mean-time-to-failure (MTTF), the ability to recover from failure, and the predictability of the program.
- **Performance:** is measured by processing speed, response time, resource consumption, throughput and efficiency.
- **Supportability:** combines the ability to extend the program extensibility, adaptability, serviceability (maintainability). In addition, testability, compatibility, configurability, etc.

Design Concepts:

1. Abstraction:

At the highest level of abstraction, a solution is stated in broad terms using the language of the problem environment. At lower levels of abstraction, a more detailed description of the solution is provided.

As we move through different levels of abstraction, we work to create procedural and data abstractions. A procedural abstraction refers to a sequence of instruction that have a specific and limited function. An example of a procedural abstraction would be the word open for the door.

A data abstraction is a named collection of data that describes a data object. In the context of the procedural abstraction open, we can define a data abstraction for door would encompass a set of attributes that describe the door (e.g. door type, swing direction, weight)

2. Architecture

Software architecture alludes to the 'overall structure of the software and the ways in which the structure provides conceptual integrity for a system'.

In its simplest from, architecture is the structure of organization of program components (modules), the manner in which these components interact, and the structure of data that are used by the components.

To goal of the software design is to derive an architectural rendering of a system. This rendering serves as a framework from which detailed design activities are constructed.

A set of architectural patterns enable a software engineer to reuse design-level concepts.

The architectural design can be represented using one or more of a number of different models.

- **Structural models** represent architecture as an organized collection of program components.
- **Framework models** increase the level of design abstraction by attempting to identity repeatable architectural design frameworks that are encountered in similar types of applications.
- **Dynamic models** address the behavioral aspects of the program architecture, indicating how the structure or system configuration may change as a function of external events.
- **Process models** focus on the design of business or technical process that the system must accommodate.
- **Functional models** can be used to represent the functional hierarchy of a system.

3.Patterns:

A design pattern 'conveys the essence of a proven design solution to a recurring problem within a certain context amidst computing concerns'.

The intent of each design pattern is to provide a description that enables a designer to determine:

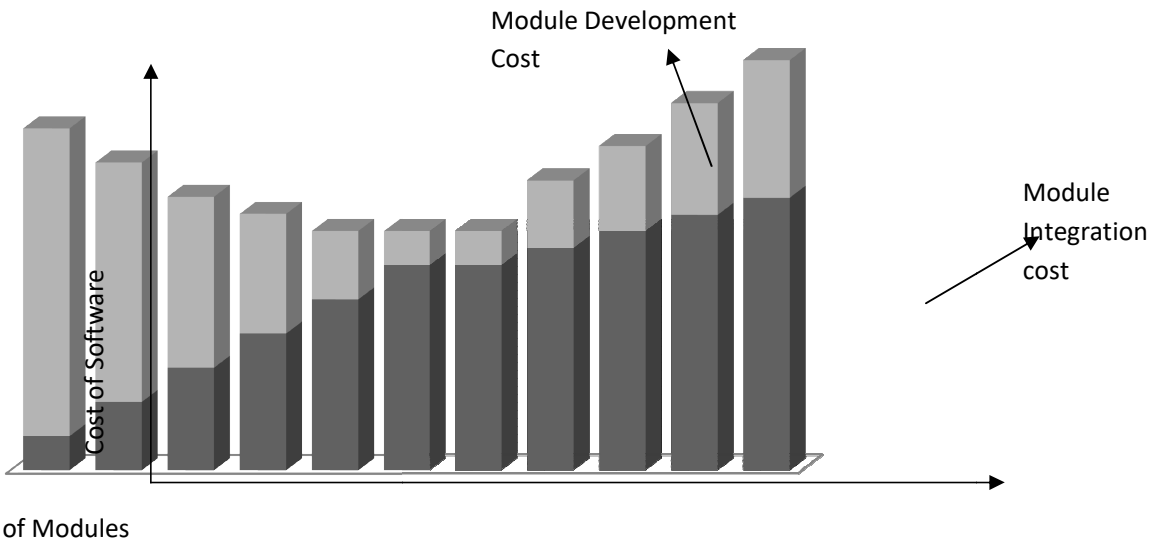
Whether the pattern is applicable to the current work, whether the pattern can be reused, and whether the pattern can serve as a guide for developing a similar, but functionally or structurally different pattern.

4. Modularity:

Software architecture and design patterns embody modularity, that is software is divided into separately names and addressable components, sometimes called modules that are integrated to satisfy problem requirements.

Monolithic software (large program composed of a single module) cannot be easily grasped by a software engineer. The number of control paths, span of reference, number of variable, and overall complexity would make undertaking close to impossible.

It is the compartmentalization of data and function. It is easier to solve a complex problem when you break it into manageable pieces 'Divide-and-conquer'. Don't over-modularize. The simplicity of each small module will be overshadowed by the complexity of integration 'cost'.



5. Information hiding

It is about controlled interfaces. Modules should be specified and design so that information (algorithm and data) contained within a module is inaccessible to other modules that have no need for such information.

Hiding implies that effective modularity can be achieved by defining by a set of independent modules that communicate with one another only that information necessary to achieve software function.

The use of information Hiding as a design criterion for modular system provides that greatest benefits when modifications are required during testing and later, during software maintenance. Because most data and procedure are hidden from other parts of the software, inadvertent errors introduced during modification are less likely to propagate to other location the software.

6. Independence

The concept of functional independence is a direct outgrowth of modularity and the concepts of abstraction and information hiding.

Design software so that each module addresses a specific sub-function of requirements and has a simple interface when viewed from other parts of the program structure. Functional independence is a key to good design, and design is the key to software quality. Independence is assessed using two qualitative criteria: Cohesion and Coupling.

Cohesion is an indication of the relative functional strength of a module.
Coupling is an indication of the relative independence among modules.

7. Refinement

It is the elaboration of detail for all abstractions. It is top down strategy. A program is developed by successfully refining level of procedural details. A hierarchy is developed by decomposing a macroscopic statement of function (a procedural abstraction) in a stepwise fashion until programming language statements are reached.

We begin with a statement of function or data that is defined at a high level of abstraction. The statement describes function or information conceptually but provides no information about the internal workings of the function or the internal structure of the data. Refinement causes the designer to elaborate on the original statement, providing more and more details as such successive refinement (elaboration) occurs.

Abstraction enables a designer to specify procedure and data and yet suppress low-level details. Refinement helps the designer to reveal low-level details as design progresses. Refinement causes the designer

to elaborate on the original statement, proving more and more details as each successive refinement 'elaboration' occurs.

8. Refactoring

It is a reorganization technique that simplifies the design of a component without changing its function or behavior. When software is refined, the existing design is examined for redundancy, unused design elements, inefficient or unnecessary algorithms, poorly constructed data structures, or any other design failure that can be corrected to yield a better design.

The Design Model creating an Architectural Design

The Design model

The design model is an abstraction of the implementation of the system. It is used to conceive as well as document the design of the software system.

Types of design elements:

1. Data Design elements:

The data design element produced a model of data that represent a high level of abstraction. This model is than more refined into more implantation specific representation which is processed by the computer based system. The structure of data is the most important part of the software design.

2. Architectural Design elements:

The architecture design elements provide us overall view of the system. The architectural design element is generally represented as a set of interconnected subsystem that are derived from analysis packages in the requirement model.

The architecture model is derived from following source:

The information about the application domain to built the software. Requirement model elements like data flow diagram or analysis classes, relationship and collaboration between them. The architectural style and pattern as per availability.

3. Interface design elements:

The interface design element for software represents the information flow within it and out of the system. They communicate between the components defined as part of architecture.

Following are the important elements of the interface design:

The user interface.

The external interface to the other system, networks etc.

The internal interface between various components.

4. Component level diagram elements

The component level design for software is similar to the set of detailed specification of each room in a house. The component level design for the software completely describes the internal details of the each software component. The processing of data structure occurs in a component and an interface which allows all the component operations.

5. Deployment level design element:

The deployment level design element shows the software functionality and subsystem that allocated in the physical computing environment which support the software.

Software Architecture

Software architecture is a structure of system which consists of various components, externally visible properties of these components and the inter-relationship among these components.

Importance of Software Architecture

The representation of software architecture allows the communication between all stakeholders and the developer.

The architecture focuses on the early design decisions that impact on all software engineering work and it is the unlimited success of the system.

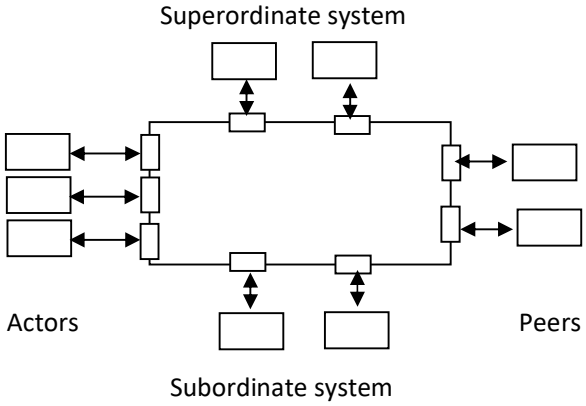
The software architecture composes a small and intellectually graspable model.

This model helps the system for integrating the components using which the components are work together.

Architectural Design:

The Architectural design starts than the developed software is put into the context. The information is obtained from the requirement model and other information collect during the requirement engineering.

Representing the system in context



All the following entities communicate with the target system though the interface that is small rectangle shown in above figure.

- **Superordinate system:** This system uses the target system like a part of some higher-level processing scheme.
- **Subordinate system:** This system is used by the target system and provides the data mandatory to complete target system functionality.
- **Peer-Level System:** This system interact on peer-to-peer basis means the information is consumed by the target system and the peers.
- **Actors**:These are the entities like people, device which interact with the target system by consuming information that is mandatory for requisite processing.

Data Design

Data design is basically the model of data that is represented at the high level of abstraction. The data design is progressively refined to create implementation specific representations.

Various Elements of data design:

Data object: The data objects are identified and relationship among various data objects can be represented using Entity Relationship Diagram of data dictionaries.

Databases: Using software design model, the data models are translated into data structures and databases at the application level.

Data warehouses: At the business level useful information is identified from various databases and the data warehouses are created.

Guidelines for data design:

- Apply systematic analysis on data.
- Identify data structure and related operations
- Establish data dictionary
- Defer the low-level design decisions until late in the design process.
- Use information hiding in the design of data structures.
- Apply a library of useful data structures and operations.
- Use a software design and programming language to support data specification and abstraction.

Architectural Style:

The architectural style is a pattern for creating the system architecture for given problem. Each style describes a system category that encompasses.

A set of components: it performs a function required by a system. (Ex. Database, computational models).

A set of connectors: it enables "Communication, coordination and cooperation".

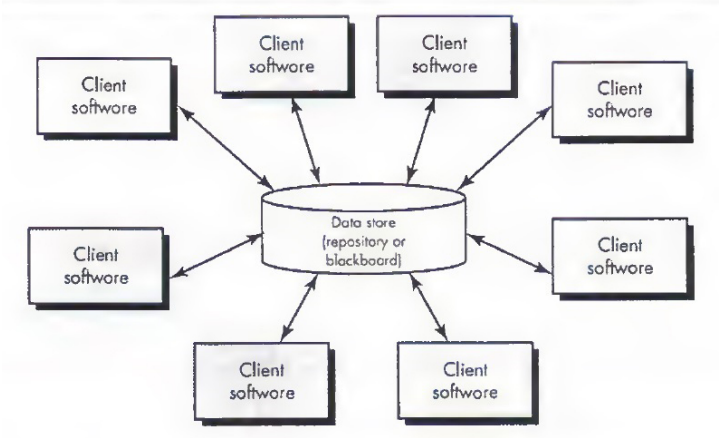
Constraints: It defines how components can be integrated to form the system.

Semantic models: It enables a designer to understand the overall properties of a system by analyzing the known properties of its constituent parts.

1. Data-Centered Architecture:

A data-centered architecture has two distinct components: a central data structure or data store (central repository) and a collection of client software.

The data store (for example, a database or a file) represents the current state of the data and the client software performs several operations like add, delete ubdate etc., on the data stored in the data store.



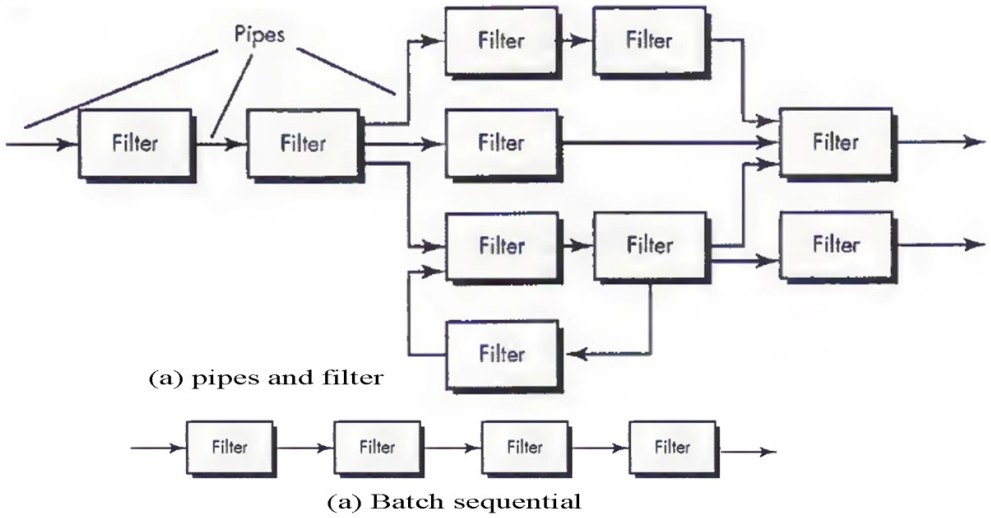
Store data is access continuously by the other components like an update, delete, add, modify from the data store. The data store allows the client software to access the data independent of any changes or the actions of other client software.

Data-centered architecture helps integrity. Pass data between clients using the blackboard mechanism. The processes are independent executed by the client components.

2. Data-flow Architecture:

Data-flow architecture is mainly used in the systems that accept some inputs and transform it into the desired outputs by applying a series of transformations.

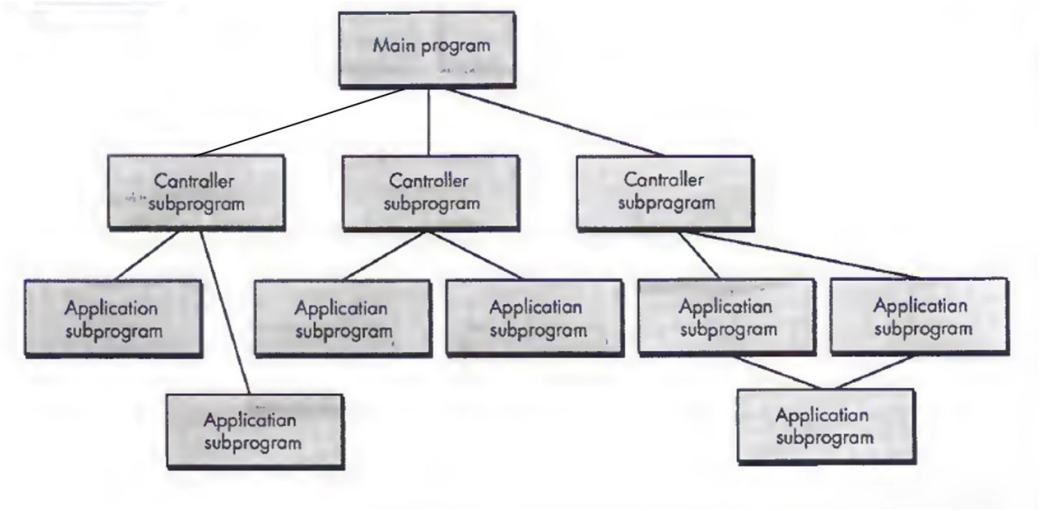
Each component, known as filter, transforms the data and sends this transformed data to other filters for further processing using the connector, known as pipe.



Each filter works as an independent entity, that is, it is not concerned with the filter which is producing or consuming the data. A pipe is a unidirectional channel which transports the data received on one end to the other end. It does not change the data in anyway: it merely supplies the data to the filter on the receiver end. The flow of data degenerates into a single line of transform then it is known as batch sequential.

3. Call and Return Architecture:

A cell and return architecture enables software designers to achieve a program structure, which can be easily modified, this style consists of the following two sub styles.



Main program/Sub-program architecture:

In this, function is decomposed into a control hierarchy where the main program involves a number of program components, which in turn may invoke other components.

Remote procedures call architecture:

In this, components of the main or subprogram architecture are distributed over a network across multiple computers.

4. Object-oriented Architecture:

In Object oriented architecture style, components of a system encapsulate data and operations, which are applied to manipulate the data. In this style, components are represented as objects they interact with each other through methods (Connectors). This architectural style has two important characteristics, which are listed below:

Objects maintain the integrity of the system.

An object is not aware of the representation of other objects.

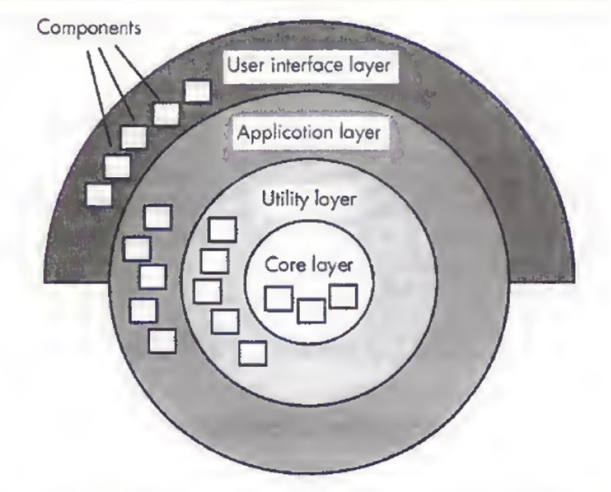
Some of the advantages associated with the object-oriented architecture are listed below:

It allows designer to decompose a problem into a collection of independent objects.

The implementation detail of objects is hidden from each other and hence, they can be changed without affecting other objects.

5. Layered Architecture:

The different layers are defined in the architecture. It consists of output and inner layer. The components of other layer manage the user interface operations.



Components execute the operating system interfacing at the inner layer. The inner layer are application layer, utility layer and the core layer. In many cases, it is possible that more than one pattern is suitable and the alternate architectural style can be designed and evaluated.

Architectural Mapping Using Data Flow Diagram:

A mapping technique, called structure design is often characterized as a data flow-oriented design method because it provides a convenient transition from a data flow diagram to software architecture. The transition from information flow to program structure is accomplished as part of a six step process.

The type of information flow is established.

Flow boundaries are indicated.

The DFD is mapped into the program structure.

Control hierarchy is defined.

The resultant structure is refined using design measures.

The architectural description is refined and elaborated.

Transform Mapping:

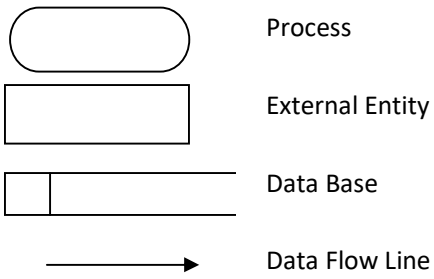
Transform mapping is a set of design steps that allows a DFD with transform flow characteristics to be mapped into a specific architectural style.

Transaction Mapping:

Transaction mapping the user interaction subsystem is considered. In transaction mapping technique the user command given as input flows into the system and produces more information flows, ultimately causes the output flow from the DFD.

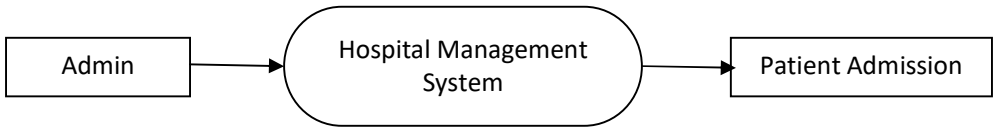
Data Flow Diagram (DFD)

Basic Components of Data Flow Diagram



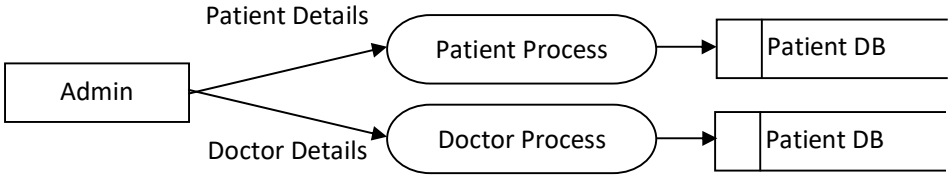
Level 0 DFD:

Review the fundamental system model. The fundamental system model can be represented by level 0 DFD and supporting information.



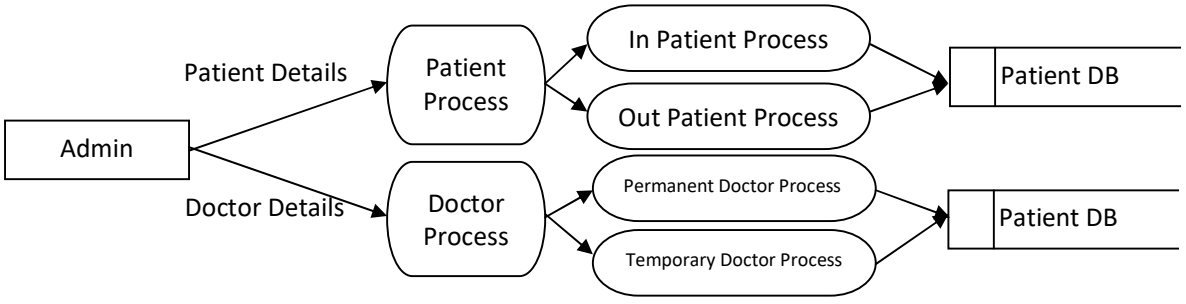
Level 1 DFD:

Review and refine data flow diagram for the software. The DFD are analyzed and refined into next higher level. Each transform in the data flow diagram impose relatively high level cohesion.



Level 2 DFD:

Determine whether the DFD has transform or transaction flow characteristics. The information flow with the system is usually represented as transform flow, There can be dominance of transaction characteristics in the DFD.



User Interface Design:

Use interface is the front-end application view to which user interacts in order to use the software. User can manipulate and control the software as well as hardware by means of user interface. User interface is part of software and is designed such a way that it is expected to provide the user insight of the software. User interface provides fundamental platform for human-computer interaction.

User interface is broadly divided into two categories:

1. Command Line Interface (CLI)
2. Graphical User Interface (GUI)

1. Command Line Interface:

CLI has been a great tool of interaction with computers until the video display monitors came into existence. CLI is first choice of many technical users and programmers. CHI is minimum interface a software can provide to its users.

CLI provides a command prompt, the place where the user types the command and feels to the system. The user needs to remember the syntax of command and its use. Earlier CLI were not programmed to handle the user errors effectively.

A command is a Text-Based reference to set of instructions, which are expected to be executed by the system. There are methods like macros, scripts that make it easy for the user to operate. A text-based command line interface can have the following elements.

- **Command Prompt**: It is text-based notified that is mostly shows the context in which the user is working. It is generated by the software system.
- **Cursor:** It is a small horizontal line or a vertical line bar of the height of line, to represent position of character while typing, Cursor is mostly found in blinking state. It moves as the user writes or deletes something.
- **Command:** A command is an executable instruction. It may have one or more parameters. Output on command execution is shows inline on the screen. When output is produced, command prompt is displayed on the next line.

2. Graphical user Interface (GUI):

Graphical user interface provides the user graphical means to interact with the system. GUI can be combination of both hardware and software. Using GUI, user interprets the software. Typically, GUI is more resource consuming than that of CLI. With advancing technology, the programmers and designers create complex GUI design that work with more efficiency, accuracy and speed.

Every graphical component provides a way to work with the system,. A GUI system has following elements such as:

- **Window:** An area where contents of application are displayed. Contents in a window ca be displayed in the form of icons or lists, if the window represents file structure. It is easier for a user to navigate in the file system in an exploring window. Windows can be minimized, resized or maximized to the size of screen.
- **Tabs:** if an application allows executing multiple instance of itself, they appear on the screen as separate windows. Tabbed interface has come up to open multiple documents in the same window. This interface also helps in viewing preference panel in application, All modern web-browsers use this feature.
- **Menu:** Menu is an array of standard commends, grouped together and placed at a visible place (usually top) inside the application window. The menu can be programmed to appear or hide on mouse clicks.
- **Icon:** An icon is small picture representing an associated application. When these icons are clicked or double clicked, the application window is opened. Icon displays application and programs installed on a system in the form of small picture.

- **Cursor:** Interacting devices such as mouse, touch pad, digital pen are represented in GUI as cursor. On screen cursor follows the instruction from hardware in almost real-tie, Cursor is also need pointers in GUI systems. They are used to select menus, windows and other applications features.

User Interface design Issues

- **Response time of the system:** Length and variability are the two important characteristics of the system response time.
- **User help facilities :** The user of each software system needs the help facility or the user manual foe the smooth use of the system.
- **Command labeling:** The commands and menu labeling must be consistent, easy to understand and learn.
- **Error information handling:** Many error messages and messages and warning are created which irritate the users as they are not meaningful. Only the critical problems should be handled.

  Error message guidelines are as follows:
  - The language of error message should be described in plain language i.e. easily understandable for the users.
  - For recovering the error, useful advice should he provided.
  - The error message must have an audible or visual indications like beep, short or the special error color.
  - The error messages must indicate any negative result so that the user verifies it.
  - The wordings of messages should not be blamed on the user.

The Golden Rules:

1. Place the user in control:

The interaction should be defined in such a way that the user is not forced to important unnecessary actions. The technical internal details must be hidden from the casual user. Design for the direct interaction with objects that appear on the screen.

2. Reduce the user's memory load:

The user interface must be designed in such a way that it reduces the demands on the user's short term memory. Create the meaningful defaults value as an advantages for the average users in the start of application. There must be a reset option for obtaining the default values. The shortcut should be easily recommended by the users.

3. Make the interface consistent:

The system must allow the user to put task into meaningful context. Consistency should be maintained for all the interaction. Do not change the past system that is created by user expectation unless there is good reason to do that.

Question and Answer - 2 – Mark

1. Define Data design?

The data design defines to transform the information domain model of analysis phase into the data structure. These structures play an important role in software implementation. The E-R Diagram and data dictionary are used to create the data design model.

2. Define Data Abstraction?

A data abstraction is a named collection of data that describes a data object. In the context of the procedural abstraction open, we can define a data abstraction for door would encompass a set of attributes that describe the door (e.g. door type, swing direction, weight).

3. Define Dynamic models?

Dynamic models address the behavioral aspects of the program architecture, indicating how the structure or system configuration may change as a function of external events.

4. Define design Pattern?

A design pattern 'conveys the essence of a proven design solution to a recurring problem within a certain context amidst computing concerns'.

5. Define Cohesion and Coupling?

Cohesion is an indication of the relative functional strength of a module.

Coupling is an indication of the relative independence among modules.

6. What is refinement?

It is the elaboration of detail for all abstractions. It is top down strategy. A program is developed by successfully refining level of procedural details. A hierarchy is developed by decomposing a macroscopic statement of function (a procedural abstraction) in a stepwise fashion until programming language statements are reached.

7. Define Model?

The design model is an abstraction of the implementation of the system. It is used to conceive as well as document the design of the software system.

8. Define architectural style?

The architectural style is a pattern for creating the system architecture for given problem. Each style describes a system category that encompasses.

A set of components: it performs a function required by a system. (Ex. Database, computational models).

A set of connectors: it enables "Communication, coordination and cooperation".

Constraints: It defines how components can be integrated to form the system.

Semantic models: It enables a designer to understand the overall properties of a system by analyzing the known properties of its constituent parts.

9. What the various elements of data design?

Data object: The data objects are identified and relationship among various data objects can be represented using Entity Relationship Diagram of data dictionaries.

Databases: Using software design model, the data models are translated into data structures and databases at the application level.

Data warehouses: At the business level useful information is identified from various databases and the data warehouses are created.

10. Define user interface?

Use interface is the front-end application view to which user interacts in order to use the software. User can manipulate and control the software as well as hardware by means of user interface.

11. Define Command Line Interface?

Command Line Interface has been a great tool of interaction with computers until the video display monitors came into existence. CLI is first choice of many technical users and programmers. CHI is minimum interface software can provide to its users.

12. Define Graphical user Interface?

Graphical user interface provides the user graphical means to interact with the system. GUI can be combination of both hardware and software. Using GUI, user interprets the software.

UNIT – IV

Testing Objectives:

1. Ensures the quality of product
2. Defect prevention and detection
3. verify and validate the user requirement
4. Ready to integration and revise the component
5. Focus on accurate and reliable result
6. Work should be performed under predefined process and software requirement and specification.
7. Discuss on generating test cases
8. Provide information to take decision for nest phase
9. Gain confidence of work
10. Evaluate the capabilities of a system and system performance.

Principals of Software Testing:

1. Process for early testing to discover defect.
2. Defect discovered analysis and correct.
3. Trying to meet user satisfaction.
4. Evaluation of user requirement.
5. Ensure product and system application.
6. It is ongoing process to achieve quality product.
7. Covers all test coverage.

Test Activities:

Test Planning: The test plan or test script is prepared. These are generated from requirements analysis document and program code.

Test case design: The goal of the test case designs to create a set of tests that are effective in testing.

Test Execution: The test data is derived through various test cases in order to obtain the test result.

Data Collection: The test results are collected and verified.

Effective evaluation: All the above test activities are performed on the software model and the maximum number of errors are uncovered.

General Characteristics of strategic Testing :

- To perform effective testing, a software team should conduct effective formal technical reviews.
- Testing begins at the component level and work outward toward the integration of the entire computer-based system.
- Different testing techniques are appropriate at different points in time.
- Testing is conducted by the developer of the software and (for large projects) by an independent test group.
- Testing and debugging are different activities, but debugging must be accommodated in any testing strategy.

Verification and Validation:

Software testing is part of a border group of activities called verification and validation that are involved in software quality assurance.

Verification refers to the set of activities that ensure that software correctly implements a specific function.

Validation refers to a different set of activities that ensure that the software that has been built traceable to customer requirements.

Verification: "Are we building the product right?"

Validation: "Are we building the right product?"

Difference between Verification and validation:

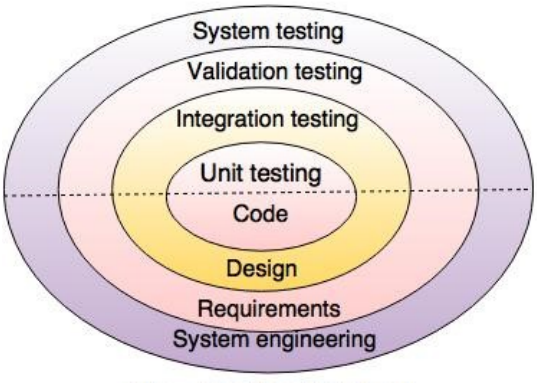
Sl. No	Verification	Validation
1.	Verification is the process to find whether the software meets the specified requirements for particular phase.	The validation process is checked whether the software meets requirements and expectation of the customer.
2.	It estimates an intermediate product.	It estimates the final product.
3.	The objectives of verification are to check whether software is constructed according to requirement and design specification.	The Objectives of the validation is to check whether the specification are correct and satisfy the business need.
4.	It describes whether the outputs are as per the inputs or not.	It explains whether they are accepted by the user or not.
5.	Verification is done before the validation.	It is done after the verification.
6.	Plans, requirement, specification, code are evaluated during the verification.	Actual product or software is tested under validation.
7.	It manually checks the files and document.	It is computer software or developed program based checking of files and document.

Software Testing Strategy:

A strategy for software testing integrates the design of software test cases into a well planned series of steps that result in successful development of the software.

Various testing Strategy:

1. **Unit Testing**: Unit testing starts at the centre and each unit is implemented in source code.
2. **Integration Testing**: An integration testing focuses on the construction and design of the software.
3. **Validation Testing**: Check all the requirements like functional, behavioral and performance requirement are validate the construction software.
4. **System Testing**: System testing confirms all system elements and performance are tested entirely.



Strategic issues

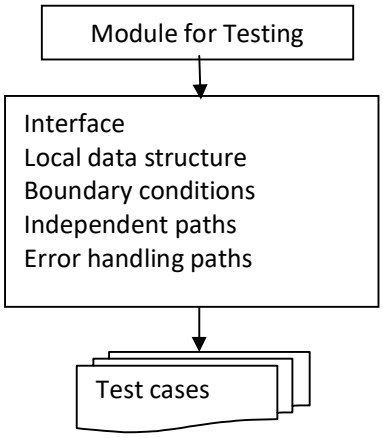
Following are the issues considered to implement software testing strategies:

- Specify the requirement before testing starts in a quantifiable manner.
- According to the categories of the user generate profile for each category of user.
- Produce robust software and it's designed to test itself.
- Should use the Formal Technical Reviews (FTR) for the effective testing.
- To access the test strategy and test cases FTR should be conducted.
- To improve the quality level of testing generates test plans from the user's feedback.

Unit Testing :

Unit testing focus on the smallest unit of software design, i.e. module or software component. Test strategy conducted on each module interface to access the flow of input and output. The local data structure is accessible to verify integrity during execution.

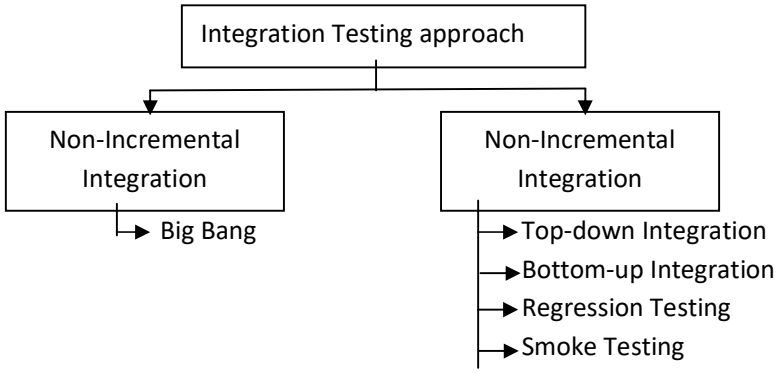
Boundary conditions are tested. In which all error handling paths are tested. An independent path is tested. The unit testing is simplified when a component with high cohesion is designed.



Integration Testing:

A group of dependent components are tasked together to ensure their quality of their integration unit. The objective is to take unit tested components and build a program structure that has been dictated by software design.

An integration testing focuses on the construction and design of the software. The integration testing can be carried out using two approaches.



Non incremental integration testing:

Combines all the components in advanced. This approach is given by the 'big-bang'. All components are combined in advance. The entire program is tested as a whole. A set of errors are tested as whole.

A set of errors is occurred then the correction is difficult because isolation cause is complex.

Incremental integration testing:

The programs are built and tested in small increments.

The errors are easier to correct and isolate.

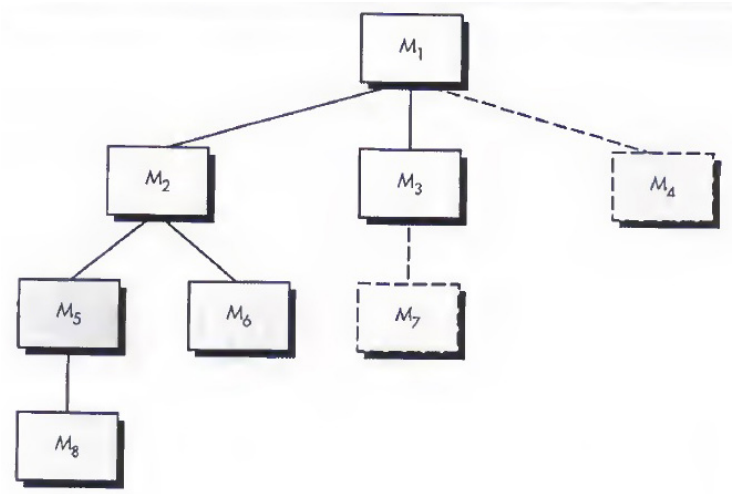
Interfaces are fully tested and applied for a systematic test approach to it.

Top-down Integration:

Top-down Integration testing is an incremental approach to construction of program structure. Modules are integrated by moving downward through the control hierarchy. Beginning with the main control module.

Following are the problems associated in an incremental integration testing approach in which the test conditions are difficult to create.

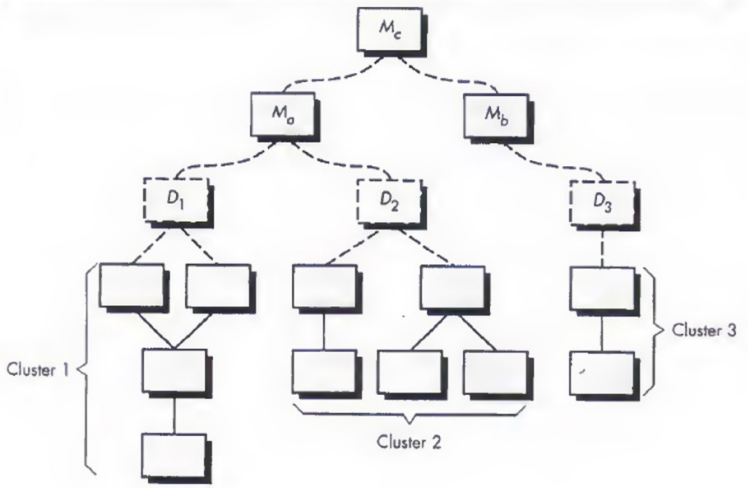
- A set of errors occur, then correction is difficult to make due to the isolation of cause.
- The programs are expanded into various modules due to the complications.
- If the previous error are corrected, then new get created and the process continues. This situation is like an infinite loop.



Bottom-Up Integration:

Bottom-up integration testing, begins construction and testing with atomic modules. Because components are integrated from the bottom-up, processing required for components subordinate to a given level is always available and the need for stubs is eliminated. The low level components are merged into clusters which perform a specific software sub function.

A control program for testing (driver) co-ordinate test case input and output. After these steps are tested in cluster. The driver is removed and clusters are merged by moving upward on the program structure.



Regression Testing:

Each time a new module is added as part of integration testing, the software changes. These changes may cause problems. In the contest of an integration test strategy, regression testing is the re-execution of some subset of tests that have already been conducted. Regression testing is the activity that helps to ensure that changes do not introduce unintended behavior or additional errors. Regression testing may be conducted manually, by re-executing a subset of all test cases.

Smoke Testing:

Smoke testing is an integration testing approach that is commonly used when 'shrink wrapped' software products are being developed, allowing the software the software team to access its project on a frequent basis. The smoke testing approach encompasses the following activities.

1. Software components that have been translated into code are integrated into a 'build'. A build includes all data files, libraries, reusable modules, and engineered components.
2. A series of tests is designed to expose errors that will keep the build from properly performing its function.
3. The build is integrated with other and the entire product is smoke tested daily. The integration approach may be top-down or bottom-up.

Acceptance Testing:

The acceptance testing is a kind of testing conducted to ensure that the software works correctly in the user environment.

Two types of acceptance testing are:

Alpha Test:

- Conducted at the developer's site by end user.
- Software is used in a natural setting with developers watching intently.
- Testing is conducted in a controller environment.

Beta Test:

- Conducted at end-user sites.
- Developer is generally not present.
- It serves as live application of the software in an environment that cannot be controlled by the developer.
- The end-user records all problems that are encountered and reports these to the developers at regular intervals.

System Testing:

System testing is known as the testing behavior of the system or software according to the software requirement specification. It is a series tests. It allows to test, verify and validate the business requirement and application architecture. The primary motive of the tests is entirely to test the computer based system.
Following are the system tests for software based systems.

1. **Recovery Testing:** To check the recovery of the software, force the software to fail in various ways. Re-initialization, check pointing mechanism, data recovery and restart are evaluated corrections.
2. **Security Testing:** It checks the system protection mechanism and secure improper penetration.
3. **Stress Testing:** System executes in a way which demands resources in abnormal quantity, frequency. A variation of stress testing is known as sensitivity testing.
4. **Performance Testing:** Performance testing is designed to test run-time performance of the system in the context of an integrated system. It always combines with the stress testing and needs both hardware and software requirements.
5. **Deployment Testing:** It is also known as configuration testing. The software works in each environment in which it is to be operated.

Testing Tactics:

White Box Testing:

White box testing (also known as Clear Box Testing, Open Box Testing, Glass Box Testing, Transparent Testing, Code-based testing or structural testing) is a software testing method in which the internal structure/ design / implementation of the item being tested.
In white box testing the test cases are derived for:

- Examine all the independent paths within a module.
- Exercise all the logical paths with their true and false sides.
- Executing all the loops within their boundaries and within operational bounds.
- Exercising internal data structures to ensure their validity.

Who perform the white box testing?

- Programmers may have some incorrect assumptions while designing or implementing some functions. Due to this there are chances of having logical errors in the program. To detect and correct such logical errors procedural details need to be examined.
- Certain assumptions on flow of control and data lead programmer to make design errors.
- There may be certain typological errors that remain undetected eve after syntax and type checking mechanism.

Advantages:

- Testing can be commenced at an earlier stage. One need not wait for the GUI to be available.
- Testing is more through, with the possibility of covering most paths.

Disadvantages:

- Since tests can be very complex, highly skilled resources are required, with a through knowledge of programming and implementation.
- Test script maintenance can be a burden if the implementation changes too frequently.
- Since this method of testing is closely tied to the application being tested, tools to cater to every kind of implementation platform may not be readily available.

White box testing methods are:

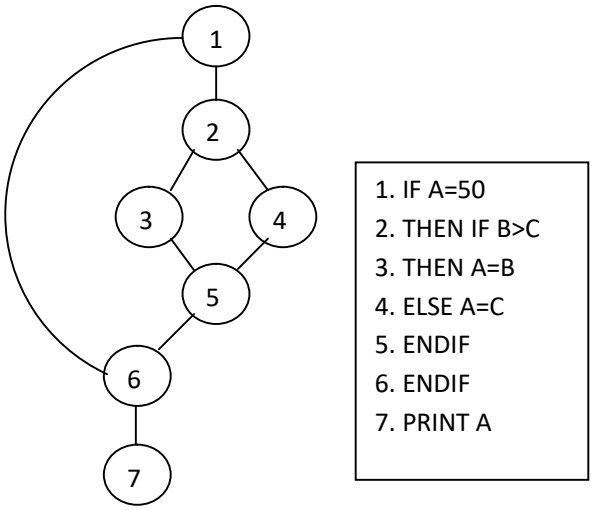
1. Basis path testing
2. Control Structure testing.

Basis path Testing:

Basis path testing is a white box testing techniques. The basis path method enables the test case designer to derive a logical complexity measure of procedural design and use this method as a guide for defining basis set of execution path.

Test cases derived to exercise the basis test are guaranteed to execute every statement in the program at least one time during testing. The objective behind basis path in software testing is that it defines the number of independent paths, thus the number of test cases needed can be defined explicitly.

Here we will take a simple example, to get a better idea what is basis path testing include.



In the above example, we can see there are few conditional statements that is executed depending on what condition it suffice. Here there are 3 paths or conditions that need to be tested to get the output.

Path 1: 1,2,3,5,6,7
Path 2: 1,2,4,5,6,7
Path 3: 1,6,7

Steps for basis path testing

- The basic steps involved in basis path testing include
- Draw a control graph (to determine different program paths).
- Calculate Cyclomatic complexity (metrics to determine the number of independent paths).
- Find the basis set of paths.
- Generate test cases to exercise each path.

Advantages of Basis path testing

- It helps to reduce the redundant tests.
- It focuses attention on program logic.
- It helps facilities analytical verses arbitrary case design.
- Test cases which exercise basis set will execute every statement in a program at least one.

Control Structure Testing:

The control structure testing is a wide testing study and also improves the quality of white-box testing.

Condition Testing:

Condition testing is a test-case design method that exercises the logical condition contained in a program module. A simple condition is a Boolean, Variable or a relational expression, possibly preceded with one NOT (-) operator. A relational expression takes the form.

Data Flow Testing:

The data testing method selects test paths of a program according to the locations of definitions and uses of various in the program. Assume that each statement in a program is assigned a unique statement number and that each function does but modify its parameter or global variables.

Loop Testing:

Loops are the cornerstone for the vast majority of all algorithms implemented in software. Loop testing is a white - box testing technique that focuses exclusively on the validity of loop constructs. Four different classes of loops can be defined. Simple Loops, Concatenated Loops, Nested Loops and Unstructured Loops.

Simple loops:

The following set of tests can be applied to simple loops, where n is the maximum number of allowable passes through the loop.

1. Skip the loop entirely.
2. Only one pass through the loop.
3. Two passes through the loop.

Nested loops:

If we were to extend the test approach for simple loops to nested loops, the number of possible tests would grow geometrically as the level of nesting increases. This would result in an impractical number of tests.

1. Start at the innermost loop. Set all other loops to minimum values.
2. Conduct simple loop tests for the innermost loop while holding the outer loops at their minimum iteration.
3. Conducting tests for the next loop, but keeping all other outer loops at minimum values and other nested loops to "typical" values.
4. Continue until all loops have been tested.

Concatenated loops:

Concatenated loops can be tested using the approach defined for simple loops, if each of the loops is independent of the other. However, if two loops are concatenated and the loop counter for loop 1 is used as the initial value for loop 2, then the loops are not independent.

Unstructured loops:

Whenever possible, this class of loops should be redesigned to reflect the use of the structured programming constructs.

BLACK BOX TESTING:

Black-box testing also called as behavioral testing. Black box testing focus on the functional requirements of the software.

This method attempts to find errors in the following categories:

- Incorrect or missing functions.
- Interface errors.
- Errors in data structures or external database access.
- Behavior or performance errors.
- Initialization and termination errors.

Techniques involved in black box testing:

Following are some techniques that can be used for designing black box tests.

Equivalence partitioning:

It is a software test design technique that involves dividing input values into valid and invalid partitions and selecting representative values from each partition as test data.

Equivalence partitioning is a black-box testing method that divides the input domain of a program into classes of data from which test cases can be derived. Test-case design for equivalence partitioning is based on an evaluation of equivalence classes for an input condition.

If a set or objects can be linked by relationships that are symmetric transitive, and reflexive, an equivalence class is present an equivalence class represents are set of valid or invalid states for input conditions.

Equivalence classes may be defined according to the following guidelines.

- If an input condition specifies a range, one valid and two invalid equivalence classes are defined.
- If an input condition requires a member a specific value, one valid and two invalid equivalence classes are defined.
- If an input condition specifies a member of a set, one valid and one invalid equivalence class are defined.
- If an input condition is Boolean, one valid and one invalid class are defined.

Boundary Value Analysis:

It is a software test design technique that involves the determination of boundaries for input values and selecting values that are at the boundaries and just inside/ outside of the boundaries as test data.

A boundary value analysis is a testing technique in which the elements at the edge of the domain are selected and tested. Using boundary value analysis instead of focusing on input conditions only, the test cased from output domain is also derived.

Guidelines for Boundary Value Analysis:

1. 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 just below a and b.
2. If an input condition specifies a number of values, test cases should be developed that exercise the minimum and maximum numbers. Values just above and below minimum and maximum are also tested.
3. Apply guidelines l and 2 to output conditions. For example, assume that a temperature versus pressure tables is required as output from an engineering analysis program. Test cases should be designed to create an output report that produces the maximum (and minimum) allowable number of table entries.

Question and Answer - 2 – Mark

1. Define Software testing

Software testing is a process of executing a source code or application with intent to identifying and eliminating bugs from the source code or application.

2. Define Black box testing.

The black box testing is used to demonstrate that the software functions are operational. As the name suggests in black box testing it is tested whether the input is accepted properly and output is correctly produced.

3. Define White box testing.

WHITE BOX TESTING (also known as Clear Box Testing, Open Box Testing, Glass Box Testing, Transparent Box Testing, Code-Based Testing or Structural Testing) is a software testing method in which the internal structure/design/implementation of the item being tested.

4. What are the different type of testing activities?

The different type of testing activities are:

Test Planning - The test plan or test script is prepared. These are generated from requirements analysis document and program code.

Test case design - The goal of test case design is to create a set of tests that are effective in testing.

Test Execution-The test data is derived through various test cases in order to obtain the test result.

Data collection - The test results are collected and verified.

Effective evaluation - All the above test activities are performed on the software model and the maximum number of errors are uncovered

5. Difference between Verification and Validation:

Sl. No	Verification	Validation
1.	Verification is the process to find whether the software meets the specified requirements for particular phase.	The validation process is checked whether the software meets requirements and expectation of the customer.
2.	It estimates an intermediate product.	It estimates the final product.
3.	The objectives of verification are to check whether software is constructed according to requirement and design specification.	The Objectives of the validation is to check whether the specification are correct and satisfy the business need.
4.	It describes whether the outputs are as per the inputs or not.	It explains whether they are accepted by the user or not.
5.	Verification is done before the validation.	It is done after the verification.
6.	Plans, requirement, specification, code are evaluated during the verification.	Actual product or software is tested under validation.
7.	It manually checks the files and document.	It is computer software or developed program based checking of files and document.

6. Define Unit Testing.

Unit testing focus on the smallest unit of software design, i.e module or software component. Test strategy conducted on each module interface to access the flow of input and output. The local data structure is accessible to verify integrity during execution.

7. Define Smoke testing

Smoke testing is an integration testing approach that is commonly used when "shrink wrapped'" software products are being developed, allowing the software team to assess its project on a frequent basis.

8. Define system testing.

System testing is known as the testing behavior of the system or software according to the software requirement specification. It is a series of various tests. It allows to test, verify and validate the business requirement and application architecture. The primary motive of the tests is entirely to test the computer-based system.

9. Define Debugging process.

Debugging process is not a testing process, but it is the result of testing. This process starts with the test cases. The debugging process gives two results, i.e. the cause is found and corrected second is the cause is not found.

10. Define Basis path testing.

Basis path testing is a white box testing technique. The basis path method enables the test case designer to derive a logical complexity measure of procedural design and use this method as a guide for defining basis set of execution path.

11. What is Cyclomatic complexity?

Cyclomatic complexity is software metric that gives the quantitative measure of logical complexity of the program.

The Cyclomatic complexity defines the number of independent paths in the basic set of program that provides the upper bound for the number of test must be conducted to ensure that all the statements have been executed at least once.

12. What is Condition Testing?

Condition testing is a test - case design method that exercises the logical conditions contained in a program module. A simple condition is a Boolean, variable or a relational expression, possibly Preceded with one NOT (-) operator. A relational expression takes the form.

13. Define boundary value analysis.

A boundary value analysis is a testing technique in which the elements at the edge of the domain are selected and tested. Using boundary value analysis instead of focusing on input conditions only, the test cased from output domain is also derived.

UNIT – V

PROJECT MANAGEMENT

Project management comprises of a number of activities, which contains planning of project, deciding scope of software product, estimation of cost in various terms, scheduling of tasks and events, and resource management.

Management Spectrum

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

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

The People:

People of a project includes from manager to developer, from customer to end user. 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).

The people capability maturity model defines the following key practice areas for software people: staffing, communication and coordination, work environment, performance management, training, compensation, competency analysis and development, career development, workgroup development, team/culture development, and others.

The Product:

The Product Before a project can be planned, 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.

Once the product objectives and scope are understood, alternative solutions are considered. The alternatives enable managers and practitioners to select a "best" approach.

The Process:

Software process provides the framework from which a comprehensive plan for software development can be established. A small number of framework activities are applicable to all software projects, regardless of their size or complexity.

A number of different tasks set - 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 such as software quality assurance, software configuration management, and measurement overlay the process model .Umbrella activities are independent of any one framework activity and occur throughout the process.

The Project:

The project is the complete software project that includes requirement analysis, development, delivery, maintenance and updates. The project manager of a project or sub-project is responsible for managing the people, product and process.

The responsibilities or activities of software project manager would be a long list but that has to be followed to avoid project failure. A software project could be extremely complex and as per the industry data the failure rate is high.

THE PEOPLE :

The most important ingredient that was successful on this project was having smart people. The most important thing you do for a project is selecting the staff. The success of the software development organization is very much associated with the ability to recruit good people.

The Stakeholders:

The software process is populated by stakeholders who can be categorized into one of five constituencies:

1. **Senior managers**- Who define the business issues that often have a significant influence' on the project?
2. **Project (technical) managers** - Who must plan, motivate, organize, and control the practitioners who do software work.
3. **Practitioners** - Who deliver the technical skills that are necessary to engineer a product or application?
4. **Customers -**who specify the requirements for the software to be engineered and other stakeholders who have a peripheral interest in the outcome.
5. **End users** - Who interact with the software once it is released for production use?

The Software Team:

Democratic decentralized (DD):

This software engineering team has no permanent leader. Rather, "task coordinators are appointed for short durations and then replaced by others who may coordinate different tasks." Decisions on problems and approach are made by group consensus. Communication among team members is horizontal.

Controlled decentralized (CD)

This software engineering team has a defined leader who coordinates specific tasks and secondary leaders that have responsibility for subtasks. Problem solving remains a group activity, but implementation of solutions is partitioned among subgroups by the team leader. Communication among subgroups and individuals is horizontal. Vertical communication along the control hierarchy also occurs.

Controlled Centralized (CC):

Top-level problem solving and internal team coordination are managed by a team leader. Communication between the leader and team members is vertical. The description of seven project factors that should be considered when planning the structure of software engineering teams:

1. The difficulty of the problem to be solved.
2. The size of the resultant program(s) in lines of code or function points
3. The time that the team will stay together (team lifetime).
4. The degree to which the problem can be modularized.
5. The required quality and reliability of the system to be built.
6. The rigidity of the delivery date.
7. The degree of sociability (communication) required for the project.

Team Leaders:

1. **Motivation** - The ability to encourage (by "push or pull technical people to produce to their best ability.
2. **Organization** - The ability to mold existing processes (or invent new ones) that will enable the initial concept to be translated into a final product.
3. **Ideas or innovation** - The ability to encourage people to create and feel creative even when they must work within bounds established for a particular software product or application. Another view of the characteristics that define an effective project manager emphasizes four key traits:
    - **Problem solving** - An effective software project manager can diagnose the technical and organizational issues that are most relevant, systematically structure a solution or properly motivate other practitioners to develop the solution, apply lessons learned from past projects to new situations, and -remain flexible enough to change direction if initial attempts at problem solution are fruitless.
    - **Managerial identity** - A good project manager must take charge of the project. They must have the confidence to assume control woes necessary and the assurance to allow good technical people to follow their instincts.

- **Achievement** -A competent manager must reward initiative and accomplishment to optimize the productivity of a project team. She must demonstrate through her own actions that controlled risk taking will not be punished.
- **Influence and team building** - An effective project manager must be able to "read" people; she must be able to understand verbal and nonverbal signals. and react to the needs of the people sending these signals. The manager must remain under control in high-stress situations.

Agile Teams :

Agile software development has been suggested as an antidote to many of the problems that have plagued software project work. To review, the agile philosophy encourages customer satisfaction and early incremental delivery of software, small highly motivated project teams.

The small, highly motivated project team, also called an agile team, adopts many of the characteristics of successful software project teams. The agile philosophy stresses individual (team member) competency coupled with group collaboration as critical success factors for the team.

"If the people on the project are good enough, they can use almost any process and accomplish their assignment. If they are not good enough, no process will repair their inadequacy-"people trump process is one way to say this.

However, lack of user and executive support can kill a project politics trump people." Inadequate support can keep even good people from accomplishing the job." Agile teams are self-organizing, a self-organizing team does not necessarily maintain a single team structure but instead uses elements of Constantine's random, open, and synchronous paradigms.

Many agile process models (e.g., Scrum) give the agile team significant autonomy to make the project management and technical decisions required to Bet the job done, Planning is kept to a minimum, and the team is allowed to select its own approach (e.g., process, methods, tools), constrained only by business requirements and organizational standards.

THE PRODUCT

A detailed analysis of software requirements would provide necessary information for estimates, but analysis often takes weeks or even months to complete. Worse, requirements may be fluid, changing regularly as the project proceeds.

Therefore we must examine the product and the problem it is intended to solve at the very beginning of the project. At a minimum, the scope of the product must be established and bounded.

Software Scope:

The first software project management activity is the determination of software scope. Scope is defined by answering the following questions:

- **Context** - How does the software to be built fit into a larger system, product, or business context, and what constraints are imposed as a result of the context?
- **Information objectives** - What customer-visible data objects are produced as output from the software? What data objects are required for input?
- **Function and performance** - What function does the software perform to transform input data into output? Are any special performance characteristics to be addressed?

Software project scope must be unambiguous and understandable at the management and technical levels. A statement of software scope must be bounded.

THE PROCESS

The framework activities that characterize the software process are applicable to all software projects. The problem is to select the process model that is appropriate for the software to be engineered by project team.

- The team must decide which process model is most appropriate for The customers who have requested the product and the people who will do the work.
- The characteristics of the product itself.
- the project environment in which the software team works.

When a process model has been selected, the team then defines a preliminary project plan based on the set of process framework activities.

Melding the Product and the Process:

Project planning begins with the melding of the product and the process. Each function to be engineered by your team must pass through the set of framework activities that have been defined for your software organization. The generic framework activities communication, planning, modeling, construction, and deployment.

Process Decomposition:

A software team should have a significant degree of flexibility in choosing the software process model that is best for the project and the software engineering tasks that populate the process model once it is chosen.

Once the process model has been chosen, the process framework is adapted to it. In every case, the generic process framework discussed earlier can be used. It will work for linear models, for iterative and incremental models, for evolutionary models, and even for concurrent or component assembly models.

The process framework is invariant and serves as the basis for all work performed by a software organization. Process decomposition commences when the project manager asks, "How do we accomplish this framework activity?" For example, a small, relatively simple project might require the following work tasks for the communication activity:

1. Develop list of clarification issues.
2. Meet with stakeholders to address clarification issues.
3. Jointly develop a statement of scope.
4. Review the statement of scope with all concerned.
5. Modify the statement of scope as required.

Consider a more complex project, which has a broader scope and more significant business impact. Such a project might require the following work tasks for the communication:

1. Review the customer request.
2. Plan and schedule a formal, facilitated meeting with all stakeholders.
3. Conduct research to specify the proposed solution and existing approaches.
4. Prepare a "working document" and an agenda for the formal meeting
5. Conduct the meeting.
6. Jointly develop mini-specs that reflect data, functional, and behavioral features of the software. Alternatively, develop use cases that describe the software from the user's point of view.
7. Review each mini-spec or use case for correctness, consistency and lack of ambiguity.
8. Assemble the mini-specs into a scoping document.
9. Review the scoping document or collection of use cases with all concerned.
10. Modify the scoping document or use cases as required.

Both projects perform the framework activity that we call communication, but the first project team performs half as many software engineering work tasks as the second.

THE PROJECT :

In order to manage a successful software project, you have to understand what can go wrong so that problems can be avoided. For this we need the following criteria's:

Software people don't understand their customer's needs.

1. The product scope is poorly defined.
2. Changes are managed poorly.
3. The chosen technology changes.
4. Business needs change [or are ill defined].
5. Deadlines are unrealistic.
6. Users are resistant.
7. Sponsorship is lost [or was never properly obtained).
8. The project team lacks people with appropriate skills.
9. Managers [and practitioners] avoid best practices and lessons learned.

How does a manager act to avoid the problems just noted?

To answer this question we need a five-part commonsense approach to software projects: Start on the right foot. This is accomplished by working hard (very hard) to understand the problem that is to be solved and then setting realistic objectives and expectations for everyone who will be involved in the project.

To maintain momentum, the project manager must provide incentives to keep turnover of personnel to an absolute minimum, the team should emphasize quality in every task it performs, and senior management should do everything possible Track progress. For a software project, progress is tracked as work products (e.g., models, source code, sets of test cases) are produced and approved (using technical reviews) as part of a quality assurance activity.

Make smart decisions. In essence, the decisions of the project manager and the software team should b to "keep it simple." Conduct a postmortem analysis. Establish a consistent mechanism for extracting lessons learned for each project. Evaluate the planned and actual schedules, collect and analyze software project metrics, get feedback from team members and customers.

ESTIMATION :

Software project management begins with a set of activities that are collectively called project planning. Before the project can begin the software team should estimate the work to be done, the resources that will be required, and the time that will elapse from start to finish Planning requires you to make an initial commitment, even though it's likely that this "commitment" will be proven wrong. Whenever estimates are made, you look into the future and accept some degree of uncertainty as a matter of course.

Estimation of resources, cost, and schedule for a software engineering effort requires experience, access to good historical information (metrics), and the courage to commit to quantitative predictions when qualitative information is all that exists.

Project complexity has a strong effect on the uncertainty inherent in planning. Project size is another important factor that can affect the accuracy and efficacy of estimates. The degree of structural uncertainty also has an effect on estimation risk.

The Project Planning Process :

The objective of software project planning is to provide framework that enables the manager to make reasonable estimates resources, cost, and schedule. In addition, estimates should attempt to define best-case and worst-case scenarios so that project outcomes can be bounded. The plan must be adapted and updated as the project proceeds.

Task Set for Project Planning:

1. Establish project scope.
2. Determine feasibility.
3. Analyze risks.
4. Define required resources.
    a. Determine required human resources.
    b. Define reusable software resources.
    c. Identify environmental resources.
5. Estimate cost and effort.
    a. Decompose the problem.
    b. Develop two or more estimates using size, function points, process tasks, or use cases.
    c. Reconcile the estimates.
6. Develop a project schedule.
    a. Establish a meaningful task set.
    b. Define a task network.
    c. Use scheduling tools to develop a time-line chart.
    d. Define schedule tracking mechanisms.

Software Scope and Feasibility :

Software scope describes the functions and features that are to be delivered to end users; the data that are input and output; the "content" that is presented to users as a consequence of using the software; and the performance, constraints, interfaces, and reliability that bound the system.

Scope is defined using one of two techniques:

1. A narrative description of software scope is developed after communication with all stakeholders.
2. A set of use cases3 is developed by end users.

RESOURCES :

The next planning task is estimation of the resources required to accomplish the software development effort. The three major categories of software engineering resources- people, reusable software components, and the development environment.

Human Resources:

The planner begins by evaluating software scope and selecting the skills required to complete development. Organizational position (e.g., manager, senior software engineer) and specialty (e.g., telecommunications, database, client-server) are specified.

For relatively small projects (a few person-months), a single individual may perform all software engineering tasks, consulting with specialists as required. For larger projects, the software team may be geographically dispersed across a number of different locations.

Reusable Software Resources:

Component-based software engineering (CBSE) emphasizes reusability that is, the creation and reuse of software building blocks. Such building blocks, often called components.
off-the-staff components, Existing software that can be acquired from a third party or from a pant project, Components arc purchased from a third party, are ready for use on the current project, and have been fully validated.

Full-experience components. Existing specifications, design, code, or test data developed for past projects that are similar to the software to be built for the current project. Members of the current software team have had full experience in the application area represented by these components.

Partial-experience components. Existing specifications, designs, code, or test data developed for past projects that are related to the software to be built for the current project but will require substantial modification. Members of the current software team have only limited experience in the application area represented by these components.

New components. Software components must be built by the software team specifically for the needs of the current project.

Environmental Resources:

The environment that supports a software project, often called the software engineering environment (SEE), incorporates hardware and software.

Hardware provides a platform that supports the tools (software) required to produce the work products that are an outcome of good software engineering practice.

When a computer-based system (incorporating specialized hardware and software) is to be engineered, the software team may require access to hardware elements being developed by other engineering teams.

SOFTWARE PROJECT ESTIMATION

Software cost and effort estimation will never be an exact science. Too many variables - human, technical, environmental, political can affect the ultimate cost of software and effort applied to develop it.

Software project estimation can be transformed from a black art to a series of systematic steps that provide estimates with acceptable risk.

To achieve reliable cost and effort estimates, a number of options arise:

1. Delay estimation until late in the project.
2. Base estimates on similar projects that have already been completed.
3. Use relatively simple decomposition techniques to generate project cost and effort estimates.
4. Use one or more empirical models for software cost and effort estimation.

The first option, is not practical. Cost estimates must be provided up-front. The second option can work reasonably well, if the current project is quite similar to past efforts and other project influences (e.g., the customer, business conditions, the software engineering environment, deadlines) are roughly equivalent.

The remaining options are viable approaches to software project estimation. Ideally, the techniques noted for each option should be applied in tandem; each used as a cross-check for the other.

EMPIRICAL ESTIMATION MODELS:

Estimation models for computer software use empirically derived formulas to predict effort as a function of LOC (line of code) or FP(function point).

- Resultant values computed for LOC or FP are entered into an estimation model.
- The empirical data for these models are derived from a limited sample of projects.
- Consequently, the models should be calibrated to reflect local software development conditions.

The COCOMO II Model :

Stands for Constructive Cost Model. Introduced by Barry Boehm in 1981 in his book "Software Engineering Economics".

Became one of the well-known and widely-used estimator models in the industry. It has evolved into a more comprehensive estimation model called COCOMO II. COCOMO II is actually a hierarchy of three estimation models.

As with all estimation models, it requires sizing information and accepts it in three forms: object points, function points, and lines of Source code.

COCOMO II is actually a hierarchy of estimation models that address the following areas:

Application composition model: Used during the early stages of software engineering when the following are important.

- Prototyping of user interfaces
- Consideration of software and system interaction.
- Assessment of performance Evaluation of technology maturity

Early design stage model: Used once requirements have been stabilized and basic software architecture has been established.

Post-architecture stage model: Used during the construction of the software.

COCOMO Cost Drivers:

Personnel Factors:

- Applications experience
- Programming language experience.
- Virtual machine experience
- Personnel capability
- Personnel experience
- Personnel continuity
- Platform experience
- Language and tool experience

Product Factors :

- Required software reliability
- Database size - Software product complexity
- Required reusability
- Documentation match to life cycle needs
- Product reliability and complexity

Platform Factors:

- ✓ Execution time constraint
- ✓ Main storage constraint
- ✓ Computer turn-around time
- ✓ Virtual machine volatility
- ✓ Platform volatility
- ✓ Platform difficulty

Project Factors:

- Use of software tools
- Use of modern programming practices
- Required development schedule
- Classified security application
- Multi-site development
- Requirements volatility

ESTIMATION FOR OBJECT-ORIENTED PROJECTS:

It is worthwhile to supplement conventional software cost estimation methods with a technique that has been designed explicitly for OO software with following approach.

1. Develop estimates using effort decomposition, FP analysis, and any other method that is applicable for conventional applications
2. Using the requirements model, develop use cases and determine a count. Recognize that the number of use cases may change as the project progresses.
3. From the requirements model, determine the number of key classes.
4. Categorize the type of interface for the application and develop a multiplier for support classes.
5. Multiply the total number of classes (key + support) by the average number of work units per class. Lorenz and Kidd suggest 15 to 20 person-days per class.
6. Cross-check the class-based estimate by multiplying the average number of work units per use case.

SPECIALIZED ESTIMATION TECHNIQUES:

Estimation for Agile Development. Estimation for agile projects uses a decomposition approach that encompasses the following steps:

1. Each user scenario (the equivalent of a mini use case created the very start of a project by end users or other stakeholders) 15 considered separately for estimation purposes.
2. The scenario is decomposed into the set of software engineering tasks that will be required to develop it.
3. The effort required for each task is estimated separately. Note: Estimation can be based on historical data, an empirical model, or experience."
4. Alternatively, the "volume" of the scenario can be estimated in LOC, FP, or some other volume-oriented measure (e.g., use-case count).
5. Estimates for each task are summed to create an estimate for the scenario.
6. Alternatively, the volume estimate for the scenario is translated into effort using historical data.
7. The effort estimates for all scenarios that are to be implemented for a given software increment are summed to develop the effort estimate for the increment.

SCHEDULING:

It is an important part of project planning activity. It involves deciding which tasks would be taken up when. The majority of projects are completed' late, if at all. A project schedule is required to ensure that required project commitments are met A schedule is required to track progress toward achieving these commitments.

Why software is delivered late:

- ❖ An unrealistic deadline
- ❖ Changing but unpredicted customer requirements
- ❖ Underestimation of efforts needed
- ❖ Risks not considered at the project start
- ❖ Unforeseen technical difficulties
- ❖ Unforeseen human difficulties
- ❖ Miscommunication among project staff
- ❖ Failure to recognize that project is falling behind Schedule

Project Scheduling:

On large projects, hundreds of small tasks must occur to accomplish a larger goal.

- Project manager's objectives.
- Define all project tasks
- Build an activity network that depicts their interdependencies.
- Identify the tasks that are critical within the activity network

Build a timeline depicting the planned and actual progress of each task Track task progress to ensure that delay is recognized "one day at a time". To do this, the schedule should allow progress to be monitored and the project to be controlled.

Software project scheduling distributes estimated effort across the planned project duration by allocating the effort to specific tasks. Scheduling for projects can be viewed from two different perspectives.

In the first view, an end-date for release of a computer based system has already been established and fixed. The software organization is constrained to distribute effort within the prescribed time frame.

In the second view, assume that rough chronological bounds have been discussed but that the end-date is set by the software engineering organization. Effort is distributed to make best use of resources and an end-date is defined after careful analysis of the software. The first view is encountered far more often that the second.

SOFTWARE QUALITY FACTORS:

Following are the software quality factors:

1. McCall's Quality factors
2. ISO 9126 Quality factors

McCall's Quality Factors

McCall's software quality factors focus on following aspect of a software product.

1. Operational characteristic of software product

- **Correctness -** A program satisfies specification and fulfills the customer requirements.
- **Reliability -** A program is expected to perform the intended function with needed precision.
- **Efficiency -** Amount of computing resources and the code required by a program to perform its functions.
- **Integrity -** The efforts are taken to handle access authorization.
- **Usability -** The efforts are needed to learn, operate, prepare input and interpret the output of a program.

2. Ability to undergo change or the product transition

- **Portability -** Transfer the program from one system (hardware or software) environment to another.
- **Reusability -** Ex tent to that a program or a part of a program is reused in other applications.
- **Interoperability -** The efforts are needed to couple one system to another.

3. Adaptability to new environment or product revision

- **Maintainability-** The efforts are needed to locate and fix an error in the program.
- **Flexibility -** The efforts are needed to modify an operational program.
- **Testability -** The effort needed to test a program to check that it performs its intended function.

ISO 9126 Quality factors:

It is developed in an attempt to recognize the quality attributes. The standard identifies the following quality attributes:

1. Functionality
2. Reliability
3. Usability
4. Efficiency
5. Maintainability
6. Portability

Software Reliability :

The probability of failure free program in a specified environment for a specified time is known as software reliability. The software application does not produce the desired output according to the requirements then the result in failure.

Measures of software reliability and availability:

A measure of reliability is Mean Time Between Failure (MTBF).

MTBF= MTTF+MTTR

Where, the MTTF and MTTR are Mean Time To Failure and Mean Time To Repair.

An alternate measure of reliability is Failures In Time(FI).

A software availability is the probability that a program is running according to the requirements at a given point in time.

Availability =[MTTF/ (MTTF + MTTR)] * 100%

The availability measure is an indirect measure of the maintainability of the software and it is more sensitive to MTTR.

Distributed Software Engineering:

In distributed system, various computers are connected in a network. The connected computers communicate with each other by passing the message in a network.

Distributed system issues:

Distributed system is more complex system than the system running on a single processor. Complexity occurs because various part of the system are managed separately as in the network.

ELEMENTS OF SOFTWARE QUALITY ASSURANCE :

Standards:

The IEEE, ISO, and other standards organizations have produced a broad array of software engineering standards and related documents. Standards may be adopted voluntarily by a software engineering organization or imposed by the customer or other stakeholders.

Reviews and audits:

Technical reviews are a quality control activity performed by software engineers for software engineers. Their intent is to uncover errors. Audits are a type of review performed by SQA personnel with the intent of ensuring that quality guidelines are being followed for software engineering work.

Testing:

Software testing is a quality control function that has one primary goal to find errors.

Error/defect collection and analysis:

The only way to improve is to measure how you're doing.

Change management:

Change is one of the most disruptive aspects of any software project. If it is not properly managed, change can lead to confusion, and confusion almost always leads to poor quality.

Education:

Every software organization wants to improve its software engineering practices. A key contributor to improvement is education of software engineers, their managers, and other stakeholders.

Vendor management:

Three categories of software are acquired from external software vendors

Security management:

With the increase in cyber crime and new government regulations regarding privacy, every software organization should institute policies that protect data at all levels, establish firewall protection for WebApps, and ensure that software has not been tampered with internally.

Safety:

Because software is almost always a pivotal component of human rated systems (e.g., automotive or aircraft applications), the impact of hidden defects can be catastrophic.

Risk management:

Although the analysis and mitigation of risk is the concern of software engineers,

Formal Technical Review

- The focus of FTR is on a work product that is requirement specification, a detailed component design, a source code listing for a component.
- The individual who has developed the work product i.e, the producer informs the project leader that the work product is complete and that a review is required.
- The project leader contacts a review leader, who evaluates the product for readiness, generates copy of product material and distributes them to two or three review members for advance preparation
- Each reviewer is expected to spend between one and two hours reviewing the product, making notes.
- The review leader also reviews the product and establish an agenda for the review meeting.
- The review meeting is attended by review leader, all reviewers and the producer.
- One of the reviewer act as a recorder, who notes down all important points discussed in the meeting.
- The meeting(FTR) is started by introducing the agenda of meeting and then the producer introduces his product. Then the producer "walkthrough" the product, the reviewers raise issues which they have prepared in advance.
- If errors are found the recorder note down.

Question and Answer - 2 – Mark

1. Define Project management.

Project management comprises of a number of activities, which contains planning of project, deciding scope of software product, estimation of cost in various terms, scheduling of tasks and events, and resource management.

2. Define Democratic decentralized (DD).

This software engineering team has no permanent leader. Rather "task coordinators are appointed for short durations and then replaced by others who may coordinate different tasks." Decisions on problems and approach are made by group consensus. Communication among team members is horizontal.

3. Define Controlled decentralized (CD).

This software engineering team has a defined leader who coordinates specific tasks and secondary leaders that have responsibility for subtasks. Problem solving remains a group activity, but implementation of solutions is partitioned among subgroups by the team leader. Communication among subgroups and individuals is horizontal. Vertical communication along the control hierarchy also occurs.

4. Define Agile.

Agile software development has been suggested as an antidote to many of the problems that have plagued software project work. To review, the agile philosophy encourages customer satisfaction and early incremental delivery of software, small highly motivated project teams.

5. What is Software scope?.

Software scope describes the functions and features that are to be delivered to end users; the data that are input and output, the "content" that is presented to users as a consequence of using the software; and the performance, constraints, interfaces, and reliability that bound the system.

6. Define Software quality.

Software quality is an effective software process applied in a way which creates a useful product and the product provides measurable value for those who produce and use it.

7. Define software reliability.

The probability of failure free program in a specified environment for a specified time is known as software reliability.

8. Define Distributed Software Engineering.

In distributed system, various computers are connected in a network. The connected computers communicate with each other by passing the message in a network.

9. Define Software Quality Assurance.

SQA should be utilized to the full extend with traceability of errors, cost efficient. There are the principles behind a highly efficient SQA procedure. Every business should use SQA and use it to the best as they can.