

UNIT I SOFTWARE PROCESS AND AGILE DEVELOPMENT	
Introduction to Software Engineering, Software Process, Perspective and Specialized Process Models -Introduction to Agility-Agile process-Extreme programming-XP Process.	
UNIT-I/ PART-A	
1	<p>What are the characteristics of software? or List the characteristics of software. (Apr 16, Nov 18)</p> <p>a)Functionality b) Reliability c) Usability d) Efficiency e) Maintainability f) Portability</p> <ul style="list-style-type: none"> • Software is engineered or developed; it is not manufactured in the classical sense. • Software doesn't wear out. • Although the industry is moving toward component based assembly, most software continues to be custom built.
2	<p>Write down the generic process framework that is applicable to any software project. (Nov 10, Apr 15)</p> <p>The following generic process framework is applicable to vast majority of software projects:</p> <ul style="list-style-type: none"> • Communication: This framework activity involves heavy communication and collaboration with the customer and encompasses requirements gathering and other related activities. • Planning: This activity establishes a plan for the software engineering work that follows. It describes the technical tasks to be conducted, the risks that are likely, the resources that will be required, the work products to be produced, and a work schedule. • Modeling: The activity encompasses the creation of models that allow the developer and the customer to better understand software requirements and the design that will achieve those requirements. • Construction: This activity combines code generation and the testing that is required to uncover errors in the code. • Deployment: The software is delivered to the customer who evaluates the delivered product and provides feedback based on the evaluation.
3	<p>List the goals of Software Engineering. (Apr 11)</p> <p>Software Engineering is the establishment and use of sound engineering principles in order to obtain economically software that is reliable and works efficiently on real machines. The goals of software engineering are:</p> <ul style="list-style-type: none"> • Software production which consists of developed programs and associated documentation. • The software product should have the essential product attributes maintainability, dependability, efficiency and acceptability. • It should also include suggestions for the process to be followed, the notations to be used, system models to be developed and rules governing these models and design guidelines.
4	<p>What are the difference levels of CMMI?</p> <ul style="list-style-type: none"> • Level 0 – Incomplete. • Level 1 – Performed. • Level 2 – Managed. • Level 3 – Defined. • Level 4 – Quantitatively Managed. • Level 5 – Optimized.
5	<p>What is Software Engineering? (Nov 13, 14, Apr 19)</p> <p>Software engineering is a discipline in which theories, methods and tools are applied to develop professional software. (Or) Software engineering is the systematic approach to develop and maintain a software product in a cost effective and efficient way.</p>

6	What are the two types of software products? (Apr 12) The two fundamental types of software product are <ul style="list-style-type: none">• Generic products: These are stand alone systems developed by organizations and sold on open market to any customer who is able to buy them.• Customized products: These are systems which are commissioned by a particular customer. A software contractor develops the software especially for that customer.							
7	Give two reasons why system engineers must understand the environment of a system. (Apr 12) <ul style="list-style-type: none">• Limited scope for rework during system development: Once some system engineering decisions have been made, they are very expensive to change. Reworking the system design to solve these problems is rarely possible. Interdisciplinary involvement: Many engineering disciplines are involved in system engineering. There is a lot of scope for misunderstanding because different engineers use different terminology and conventions.							
8	What is RAD? Rapid Application Development is an incremental software development process. When tools are interpreted so that information created by one tool can be used by another, a system for the support of software development called computer aided software engineering is established.							
9	List out evolutionary software process model. <ul style="list-style-type: none">• Incremental model• Spiral model• WINWIN spiral model• Concurrent development model• Object oriented model• Embedded model							
10	What are the difference between product and process? <table><tr><th>PROCESS</th><th>PRODUCT</th></tr><tr><td>It is a frame work which has a set of rules to be followed in key processing areas (KPA),rules for framing task sets, setting a milestone for it and applying s/w quality assurance points.</td><td>It is the final shipment outcome of the process.</td></tr><tr><td>It is used to obtain quality product.</td><td>Various process paradigms/models are used to build a quality product.</td></tr></table>		PROCESS	PRODUCT	It is a frame work which has a set of rules to be followed in key processing areas (KPA),rules for framing task sets, setting a milestone for it and applying s/w quality assurance points.	It is the final shipment outcome of the process.	It is used to obtain quality product.	Various process paradigms/models are used to build a quality product.
PROCESS	PRODUCT							
It is a frame work which has a set of rules to be followed in key processing areas (KPA),rules for framing task sets, setting a milestone for it and applying s/w quality assurance points.	It is the final shipment outcome of the process.							
It is used to obtain quality product.	Various process paradigms/models are used to build a quality product.							
11	What is CPF? A Common Process Framework is established by defining a small number of framework activities that are applicable to all software projects, regardless of their size or complexity.							
12	Define Computer based system and specify its components. A set of arrangement of elements that are organized to accomplish some predefined goals by processing information. Its components are software, hardware, people, database, documentation and procedure.							
13	What are the advantages and disadvantage of Incremental Model? <u>Advantages:</u> The software development activities are repeated each time there is a new release of software. It provides a platform for evaluation by the user. It can be planned to manage technical risks. It enables partial functionality to be delivered to end users without ordinary delay. <u>Disadvantage:</u> It makes the unrealistic assumptions that system as well as software requirements remain stable which is not true.							

14	<p>What are the advantages and disadvantages of Waterfall Model?</p> <p>Advantages: It provides a template into which methods for analysis, design, and other phases can be placed. It provides for baseline management. It is better than any haphazard approach to software development.</p> <p>Disadvantages: It lacks the perception for a reverse engineering on how to engineer an existing legacy system. The client has to wait until the installation and checkout phase to see how a system works. Thus a complex system requires considerable time and effort. There is no rapid prototyping and incremental development. Real time software cannot follow this model. Customer satisfaction is not full filled.</p>
15	<p>What are the advantages and disadvantages of Spiral Model?</p> <p>Advantages: It provides the potential for rapid development for incremental versions of the software. It can be applied throughout the life of the computer software. It allows the developer to apply the prototyping approach at any stage. It demands a direct consideration of technical risks at all stages.</p> <p>Disadvantages: It may be difficult to convince the customers at times especially in contrast solutions. It demands considerable risks assessment expertise and relies on them for success. It takes time for determining the efficacy and thus the model cannot be used as widely as others.</p>
16	<p>What are the advantages and disadvantage of WINWIN SPIRAL MODEL?</p> <p>Advantages: It has a provision for system stakeholders to negotiate mutually satisfactory specifications. Customer satisfaction is fulfilled. It overcame the problem of lack of anchor points to correlate the completion of the spiral cycles and organization major milestones.</p> <p>Disadvantage: The model does not specifically address the issues of how developers specify, design and test the conceptual construct software.</p>
17	<p>What are the advantages and disadvantage of Object Oriented Model?</p> <p>Advantages: Object oriented concepts like encapsulation can be improvised in this model. It simplifies software development because it hides complexity. Reusability is enhanced.</p> <p>Disadvantage: In safety critical conditions they require a design by contract in the construction of reliability software.</p>
18	<p>Give the restraining factors that are to be considered to construct a system model.</p> <ul style="list-style-type: none"> • Assumptions that reduce the number of possible permutations and variations thus enabling the model to reflect the problem in a reasonable manner • Simplifications that enable the model to be created in a timely manner, Limitations that help to bound the system • Constraints that will guide the manner in which the model is created and the approach taken when the model is implemented. • Preferences that indicate the preferred architecture for all data, functions, and technology.
19	<p>What does a System Engineering Model accomplish?</p> <ul style="list-style-type: none"> • Define processes that serve needs of view. • Represent behavior of process and assumption. • Explicitly define Exogenous and Endogenous Input. • Represent all Linkages that enable engineer to better understand view.
20	<p>What are the guidelines for Project Scheduling?(Apr 15)</p> <ul style="list-style-type: none"> • Compartmentalization. • Interdependency. • Time allocation. • Effort validation.

21	<p>What are the different considerations that have to be followed for project estimation?</p> <p>To achieve reliable cost and effort estimation, the following options are considered,</p> <ul style="list-style-type: none"> • Delay estimation until late in the project. • Base estimates on similar projects that have already been completed. • Use relatively simple decomposition techniques to generate project cost and effort estimates. • Use one or more empirical models for software cost and effort estimation.
22	<p>Define Process-based estimation. State the advantages and disadvantages in LOC based Cost Estimation. (Apr 15)</p> <p>The process is decomposed into a relatively small set of tasks and the effort required to accomplish each task is estimated.</p> <p>Advantages of LOC:</p> <ul style="list-style-type: none"> • Simple to measure. • An intuitive metric for measuring the size of software due to the fact that it can be seen and the effect of it can be visualized. <p>Disadvantages of LOC:</p> <ul style="list-style-type: none"> • Difficult to measure LOC in the early stages of a new product. • Source instructions vary with coding languages, design methods and with programmer's ability. • No industry standard for measuring LOC. • LOC cannot be used for normalizing if platforms and languages are different. The only way to predict LOC for a new app to be developed is through analogy based on similar software application. • Programmers may be rewarded for writing more LOC based on a misconception of higher management by thinking that more the LOC, means more the productivity of the programmer.
23	<p>What are the different hierarchies of estimation models?</p> <ul style="list-style-type: none"> • Application Composition Models • Early Design Stage Models. • Post-Architecture Stage Models.
24	<p>What are the advantages and disadvantages of Prototyping Model?</p> <p>Advantages:</p> <ul style="list-style-type: none"> • It produces the products quickly and thus saves the time It and solves the waiting problem in waterfall model. • It minimizes the cost and product failure. • It is possible for the developers and client to check the function of preliminary implementations of system models before committing to a final system. • It obtains feedback from clients and changes in system concept. <p>Disadvantages: It ignores quality, reliability maintainability and safety requirements. Customer satisfaction is not achieved.</p>
25	<p>What are the characteristics of Risks?</p> <ul style="list-style-type: none"> • Uncertainty – the risk may or may not happen • Loss – if the risk becomes reality, unwanted consequences or losses will occur.
26	<p>Write the IEEE definition of software engineering.(Nov 17)</p> <p>Software Engineering: The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software.</p>

27	<p>If you have to develop a word processing software product, what process model will you choose? Justify your answer. (Nov 16)</p> <p>Incremental process model is used to develop a word processing software product.</p> <p>Justification:</p> <ul style="list-style-type: none"> • Word-processing software developed using the incremental paradigm might deliver <ul style="list-style-type: none"> ✓ Basic file management, editing, and document production functions in the first increment; ✓ More sophisticated editing and document production capabilities in the second increment; ✓ Spelling and grammar checking in the third increment; ✓ Advanced page layout capability in the fourth increment.
28	<p>Depict the relationship between work product, task, activity and System. (Nov 16, Apr 17)</p> <ul style="list-style-type: none"> • A task focuses on a small, but well-defined objective that produces a tangible work product. • Work Product is any deliverable or outcome of the set of activities. • An activity is a major unit of work (set of tasks) that has to be completed to produce an outcome. • System is a set of interconnected components that carries out a specific activity.
29.	<p>What led to the transition from product oriented development to process oriented development?(Apr 16)</p> <p>The product-oriented approach to the teaching of writing emphasizes mechanical aspects of writing, such as focusing on grammatical and syntactical structures and imitating models. This approach is primarily concerned with "correctness" and form of the final product.</p> <p>Process-oriented approaches concern the process of how ideas are developed and formulated in writing. Writing is considered a process through which meaning is created. This approach characterizes writing as following a number of processes:</p> <ul style="list-style-type: none"> • First, a writer starts writing ideas as drafts. • Subsequently, he checks to see whether the writing and the organization makes sense to him or not. • After that, he checks whether the writing will be clear to the reader. • This approach focuses on how clearly and efficiently a user can express and organize his ideas, not on correctness of form.
30.	<p>Why LOC is not a better metrics to estimate a software?(Nov 17)</p> <ul style="list-style-type: none"> • Difficult to measure LOC in the early stages of a new product. • Source instructions vary with coding languages, design methods and with programmer's ability. • No industry standard for measuring LOC. • LOC cannot be used for normalizing if platforms and languages are different. The only way to predict LOC for a new app to be developed is through analogy based on similar software application. <p>Programmers may be rewarded for writing more LOC based on a misconception of higher management by thinking that more the LOC, means more the productivity of the programmer.</p>
31.	<p>What is Extreme Programming?</p> <p>Extreme programming, requirements are expressed as scenarios (called user stories), which are implemented directly as a series of tasks. Programmers work in pairs and develop tests for each task before writing the code. All tests must be successfully executed when new code is integrated into the system. There is a short time gap between releases of the system.</p>

32.	<p>If you have to develop a word processing software product, what process model will you choose? Justify your answer. (Apr 18)</p> <p>Incremental development method- different components can be developed and added incrementally.</p>
33	<p>What is Software? List its Characteristics. (Apr 18)</p> <p>Software is nothing but a collection of computer programs that are related documents that are indented to provide desired features, functionalities and better performance.</p> <p>Features:</p> <ul style="list-style-type: none"> • Software is engineered, not manufactured • Software does not wear out. <p>Most software is custom built rather than being assembled from components</p>
34	<p>What are the key features of testing in XP?</p> <p>The key features of testing in XP are:</p> <ul style="list-style-type: none"> • Test-first development, • incremental test development from scenarios, • user involvement in the test development and validation, and the use of automated testing frameworks.
35	<p>Briefly write about Agile methods.</p> <p>The best way to achieve better software was through careful project planning, formalized quality assurance, the use of analysis and design methods supported by CASE tools, and controlled and rigorous software development processes. This view came from the software engineering community that was responsible for developing large, long lived software systems such as aerospace and government systems. Teams were often geographically dispersed and worked on the software for long periods of time. An example of this type of software is the control systems for a modern aircraft, which might take up to 10 years from initial specification to deployment. These plan driven approaches involve a significant overhead in planning, designing, and documenting the system. This overhead is justified when the work of multiple development teams has to be coordinated, when the system is a critical system, and when many different people will be involved in maintaining the software over its lifetime.</p>
36	<p>Name the Umbrella activities in software process. (Nov 18)</p> <ul style="list-style-type: none"> • Communication • Planning • Modelling • Construction • Deployment
37	<p>List any two agile Process Models (Apr 19)</p> <ul style="list-style-type: none"> • Extreme Programming (XP) • Adaptive Software Development (ASD) • Dynamic Systems Development Method (DSDM) • Scrum. • Crystal. • Feature Driven Development (FDD) • Agile Modeling (AM)
38	<p>What are the various categories of software ? (Apr 21)</p> <p>Application software (application software: office suites, word processors, spreadsheets, etc.)</p> <p>System software (system software: operating systems, device drivers, desktop environments, etc.)</p> <p>Computer programming tools (programming tools: assemblers, compilers, linkers, etc.)</p>

39	List out the goals of software engineering (Apr 21) Readable, Correct, Reliable, Reusable, Extendable, Flexible, Efficient.
----	---

UNIT-I / PART-B

1. Explain the following: (i) waterfall model (ii) Spiral model (iii) RAD model
(iv) Prototyping model (*Nov 10, 13, 17, Apr 14, 15, 16, 19, 21*)

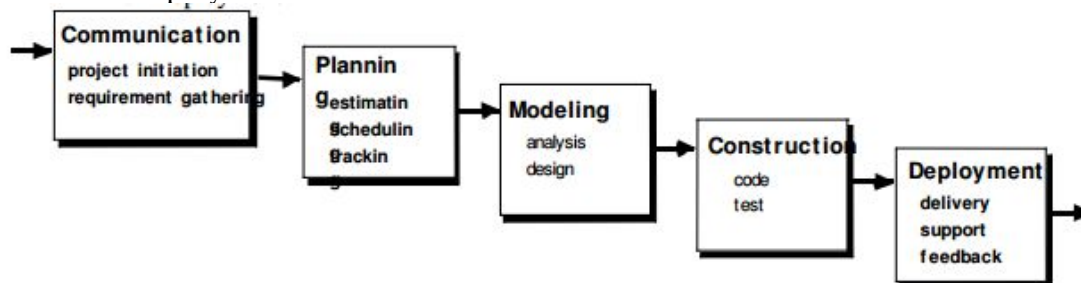
WATERFALL MODEL

When the requirements of a problem are reasonably well understood-when work flows from communication through deployment in a reasonably linear fashion. This situation is encountered when

- Well defined adaptations or enhancements to an existing system must be made.
- In limited number of new development efforts, but only when requirements are well defined and reasonably stable.

The waterfall model, called the classic life cycle suggests a systematic, sequential approach to software development.

- Begins with customer specification of requirements
- Progresses through planning
- Modeling
- Construction
- Deployment



THE WATERFALL MODEL

Problems encountered when Waterfall Model is applied:

- Real projects rarely follow the sequential flow that the model proposes. Although the linear model can accommodate iteration, it does so indirectly. As a result, changes can cause confusion as the project team proceeds.
- It is often difficult for the customer to state all the requirements explicitly. The waterfall model requires this and has difficulty accommodating uncertainty that exists at the beginning of many projects.
- The customer must have patience. A working version of the program will not be available until late in the project life-span. A major blunder, if undetected until the working program is reviewed can be disastrous.

Linear structure of the waterfall model leads to blocking states in which some project team members must wait for other members of the team to complete dependent tasks. Blocking states are more prevalent at the beginning and end of the linear sequential process.

SPIRAL MODEL

The spiral model is an evolutionary software process model that couples the iterative nature of prototyping with the controlled and systematic aspects of the waterfall model. It provides the potential for rapid development of increasingly more complex versions of the software. Boehm definition:

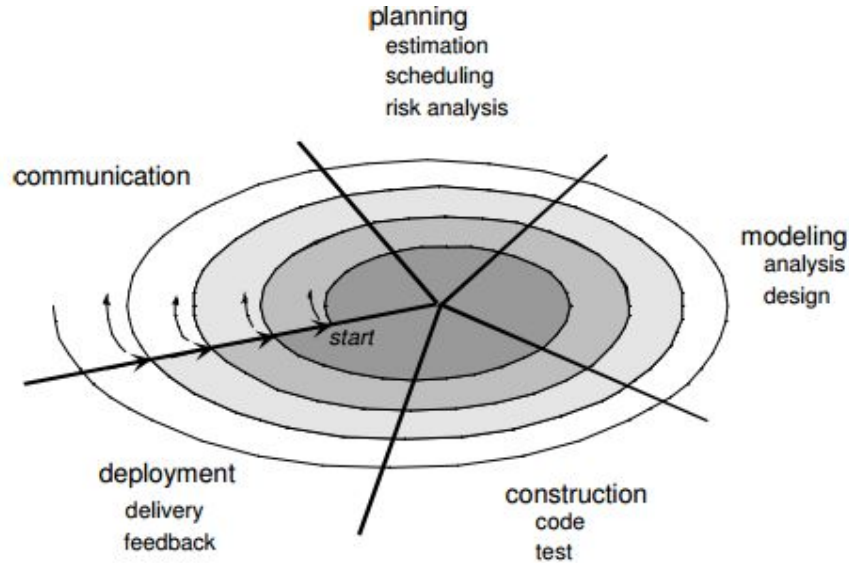
The spiral development model is a risk-driven process model generator. It has two main distinguishing features :

- One is a cyclic approach for incrementally growing a system's degree of definition

and implementation while decreasing its degree of risk.

- The other is a set of anchor point milestones for ensuring stakeholder commitment to feasible and mutually satisfactory system solutions.

Using spiral model, software is developed in a series of evolutionary releases. During early iterations, the release is a paper model or prototype. During later iterations, more complex versions of the engineered system are produced.



A TYPICAL SPIRAL MODEL

A spiral model is divided into a set of framework activities defined by the software engineering team. Each of the framework activities represents one segment of the spiral path as shown in the figure.

- As evolution begins the software team performs activities implied by the by the circuit around the spiral, in clockwise direction, beginning at the center.
- Risk is considered at each revolution made.
- Anchor point milestones-a combination of work products and conditions that are attained along the path of the spiral are noted for each evolutionary pass.
- First circuit around the spiral results in the development of product specifications.
- Subsequent passes develop a prototype, progressively lead to sophisticated version of the software.
- Cost and schedule are adjusted based on feedback derived from the customer after delivery.
- Unlike other process models that end when the software is delivered, the spiral model can be adapted throughout the life of the software.
- The first spiral represents a Concept Development Project which starts at core and continues for multiple iterations until concept development is implemented.
- The concept developed to an actual product, proceeds outward on the spiral and a New Product Development Project commences.
- The new product may evolve to represent Product Enhancement Project.

Features of Spiral Model:

- Realistic approach to the development of large scale-systems and software.
- Because the software evolves as the process progresses, the developer and customer better understand and react to risks at each evolutionary level.
- Enables the developer to apply the prototyping approach at any stage in the evolution

of the product.

- It maintains the systematic stepwise approach of classic life cycle but in incorporates it into an iterative framework that realistically reflects the real world.

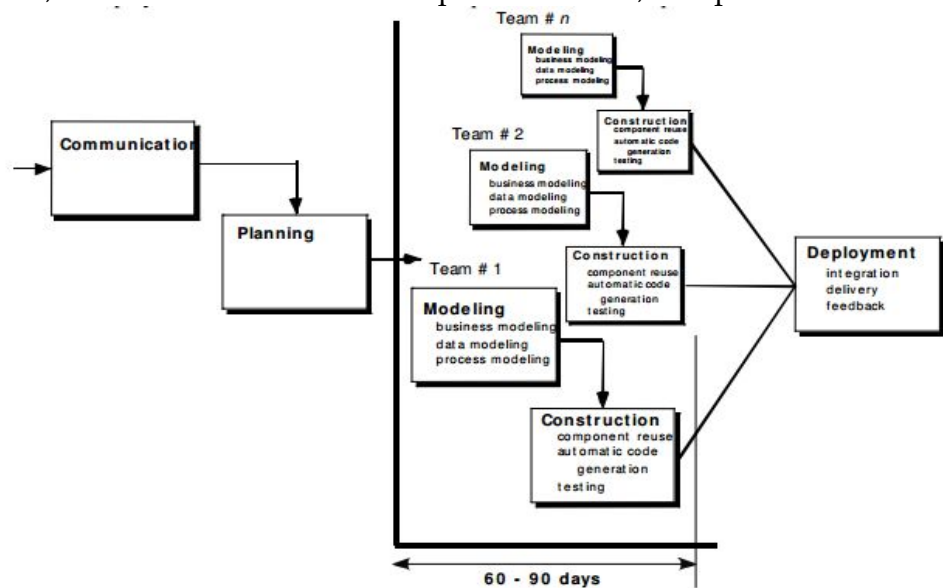
It demands direct consideration of technical risks at all stages of the project and if applied properly, reduce risks before they become problematic.

RAD MODEL

Rapid Application Development (RAD) is an incremental software process model that emphasizes a short development cycle. The RAD model is a high-speed adaptation of the waterfall model, in which rapid development is achieved by using component-based construction approach. If requirements are well understood and project scope is constrained, the RAD process enables a development team to create a fully functional system within a very short time period.

Like other process models, RAD approach maps to the generic framework activities as

- Communication, works to understand the business problem and the information characteristics that the software must accommodate.
- Planning, is essential because multiple software teams work in parallel on different system functions.
- Modeling encompasses three major phases-business modeling, data modeling and process modeling- and establishes design representations that serve as the basis for RAD's construction activity.
- Construction emphasizes the use of pre-existing software components and the application of automatic code generation.
- Deployment, establishes a basis for subsequent iterations, if required.



THE RAD MODEL

If a business application can be modularized in a way that enables each major function to be completed in less than three months, it is a candidate for RAD. Each major function can be addressed by a separate RAD team and then integrated to form a whole.

RAD Drawbacks:

- For large, but scalable projects, RAD requires sufficient human resources to create the right number of RAD teams.
- If developers and customers are not committed to the rapid fire activities, RAD project fails.
- If a system is not properly modularized, building the components for RAD will be

problematic.

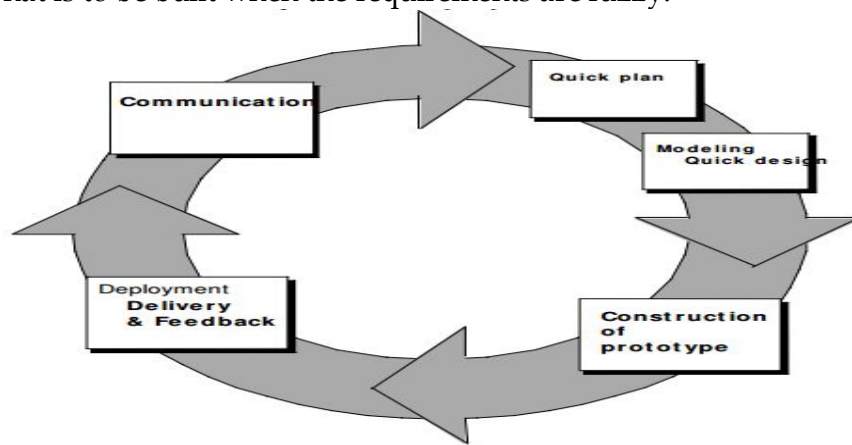
- If high performance is an issue and performance is to be achieved through tuning the interfaces to system components, the RAD approach may not work.
- Not appropriate when technical risks are high or when new technology is used.

PROTOTYPING MODEL

The customer usually defines a set of general objectives for software, but does not identify detailed input, processing or output requirements. The developer may be unsure of

- The efficiency of an algorithm
- The adaptability of an operating system
- The form that human-machine interaction should take

In these and many other situations, a prototyping paradigm may offer the best approach. Although prototyping can be used as a standalone process model, it is more often used as a technique that can be implemented within the context of any one of the other process models. The prototyping paradigm assists the software engineer and the customer to better understand what is to be built when the requirements are fuzzy.



THE PROTOTYPING MODEL

- ✓ The prototyping paradigm begins with communication. The software engineer and the customer meet and
 - Define the overall objectives for the software
 - Outline areas whether further definition is mandatory
 - Identify whatever requirements are known
- ✓ The quick design leads to construction of a prototype.
- ✓ The prototype is deployed and then evaluated by customer/user. The feedback is used to refine requirements for the software.
- ✓ Iteration occurs as the prototype is tuned to satisfy the needs of the customer, while at the same time enabling the developer to better understand what needs to be done.
- ✓ Prototyping serves as a mechanism for identifying software requirements. If a working prototype is built, the developer attempts to make use of existing program fragments or applies tools that enable working programs to be generated quickly.
- ✓ The prototype can serve as the first system. Both customers and developers like the prototyping paradigm. Users get a feel for the actual system and developers get to build something immediately.

Problems with Prototyping

- ✓ The customer sees what appears to be a working version of the software, unaware that in the rush to get it working, the quality or maintainability is not considered. When informed that the software has to be rebuilt, demands for a few fixes to make it a working product.

The developer often makes implementation compromises to make the prototype work quickly.

An inappropriate OS or programming language is used, but the developer forgets why these choices are inappropriate and gets comfortable with the system. The less-than-ideal choice has now become an integral part of the system.

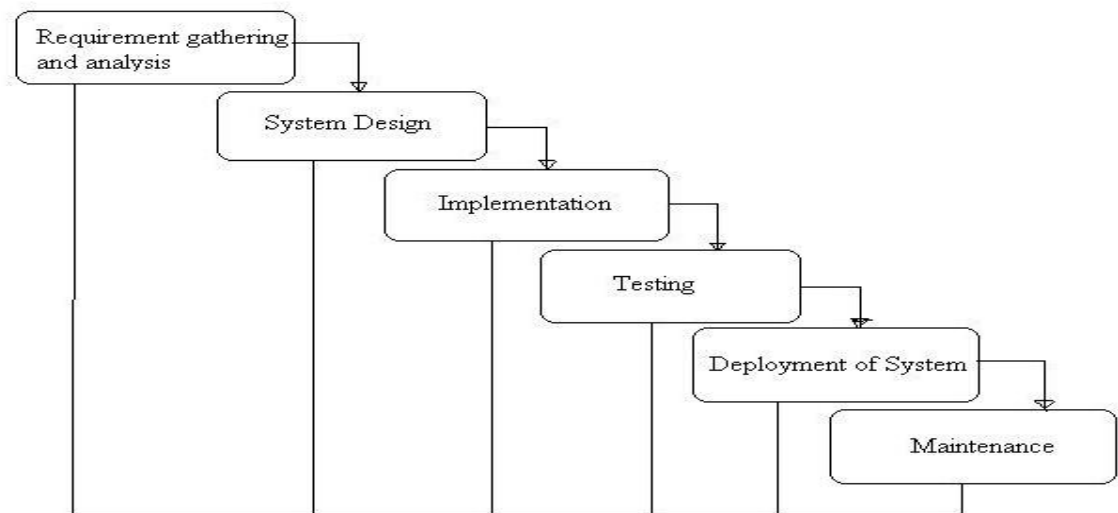
2. Describe waterfall, incremental and iterative models based SLCS and compare. (Nov 12)

The Waterfall Model

The **waterfall model** is a sequential design process, used in software development processes, in which progress is seen as flowing steadily downwards (like a waterfall) through the phases of Conception, Initiation, Analysis, Design, Construction, Testing, Production/Implementation and Maintenance. The following phases are followed in order:

- Requirements specification resulting in the product requirements document
- Design resulting in the software architecture
- Construction (implementation or coding) resulting in the actual software
- Integration
- Testing and debugging
- Installation
- Maintenance

General Overview of "Waterfall Model"



Advantages:

- Easy to understand and implement.
- Widely used and known (in theory!).
- Reinforces good habits: define-before- design, design-before-code.
- Identifies deliverables and milestones.
- Document driven, URD, SRD,. etc. Published documentation standards,
- Works well on mature products and weak teams.

Disadvantages:

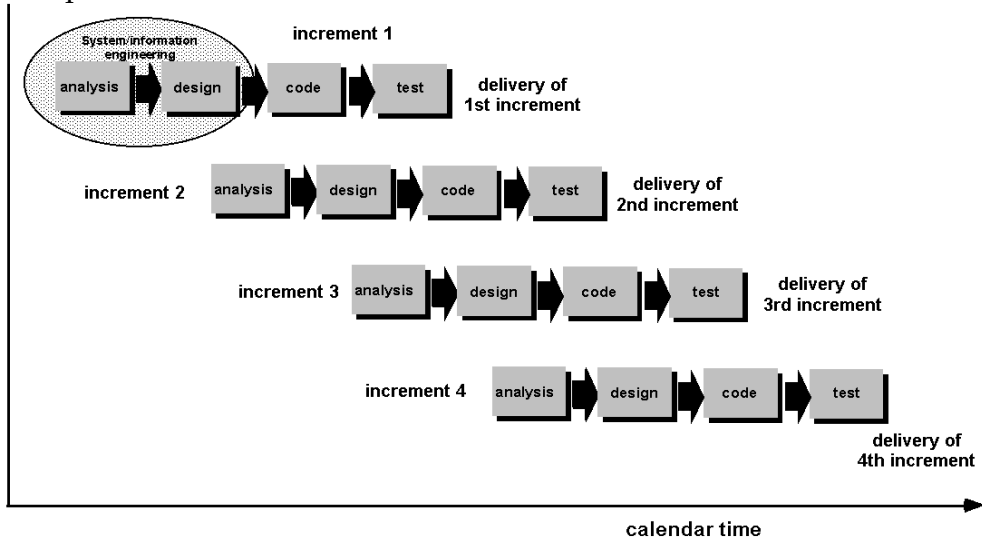
- Idealized, doesn't match reality well.
- Doesn't reflect iterative nature of exploratory development.
- Unrealistic to expect accurate requirements so early in project.
- Software is delivered late in project, delays discovery of serious errors.
- Difficult to integrate risk management.
- Difficult and expensive to make changes to documents, "swimming upstream".

- Significant administrative overhead, costly for small teams and projects

Incremental models

The incremental model applies linear sequences in a staggered fashion as calendar time progresses. Each linear sequence produces deliverable “increments” of the software in a manner that is similar to the increments produced by an evolutionary process flow.

- When an incremental model is used, the first increment is often a core product.
- The core product is used by the customer (or undergoes detailed evaluation).
- As a result of use and/or evaluation, a plan is developed for the next increment. The plan addresses the modification of the core product to better meet the needs of the customer and the delivery of additional features and functionality.
- This process is repeated following the delivery of each increment, until the complete product is produced.



Advantages of Incremental model:

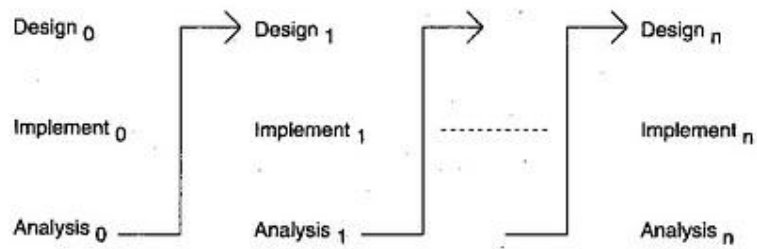
- Generates working software quickly and early during the software life cycle.
- This model is more flexible – less costly to change scope and requirements.
- It is easier to test and debug during a smaller iteration.
- In this model customer can respond to each built.
- Lowers initial delivery cost.
- Easier to manage risk because risky pieces are identified and handled during it'd iteration.

Disadvantages of Incremental model:

- Needs good planning and design.
- Needs a clear and complete definition of the whole system before it can be broken down and built incrementally.
- Total cost is higher than waterfall.

Iterative models

An iterative life cycle model does not attempt to start with a full specification of requirements. Instead, development begins by specifying and implementing just part of the software, which can then be reviewed in order to identify further requirements. This process is then repeated, producing a new version of the software for each cycle of the model.



Advantages of Iterative model:

- In iterative model we can only create a high-level design of the application before we actually begin to build the product and define the design solution for the entire product. Later on we can design and build a skeleton version of that, and then evolved the design based on what had been built.
- In iterative model we are building and improving the product step by step. Hence we can track the defects at early stages. This avoids the downward flow of the defects.
- In iterative model we can get the reliable user feedback. When presenting sketches and blueprints of the product to users for their feedback, we are effectively asking them to imagine how the product will work.
- In iterative model less time is spent on documenting and more time is given for designing.

Disadvantages of Iterative model:

- Each phase of an iteration is rigid with no overlaps
- Costly system architecture or design issues may arise because not all requirements are gathered up front for the entire lifecycle

3. Discuss in detail the project structure and programming team structure of a software organization. (Nov 10)

A PROCESS FRAMEWORK

A process framework establishes the foundation for a complete software process by identifying a small number of framework activities that are applicable to all software projects, regardless of their size or complexity. In addition, the process framework encompasses a set of umbrella activities that are applicable across the entire software process.

Each framework activity is populated by a set of software engineering actions—a collection of related tasks that produces a major software engineering work product. Each action is populated with individual work tasks that accomplish some part of the work implied by the action.

The following generic process framework is applicable to vast majority of software projects:

- **Communication:** this framework activity involves heavy communication and collaboration with the customer and encompasses requirements gathering and other related activities.
- **Planning:** this activity establishes a plan for the software engineering work that follows. It describes the technical tasks to be conducted, the risks that are likely, the resources that will be required, the work products to be produced, and a work schedule.
- **Modeling:** the activity encompasses the creation of models that allow the developer and the customer to better understand software requirements and the design that will achieve those requirements.
- **Construction:** this activity combines code generation and the testing that is required to uncover errors in the code.

- **Deployment:** the software is delivered to the customer who evaluates the delivered product and provides feedback based on the evaluation.

The modeling activity is composed of two software engineering actions:

- **Analysis** encompasses a set of work tasks requirements gathering, elaboration, negotiation, specification and validation that lead to the creation of the analysis model or requirements specification.
- **Design** encompasses work tasks data design, architectural design, interface design and component-level design and create a design model or design specification.

Each software engineering action is represented by a number of different task sets-each a collection of software engineering work tasks, related work products, quality assurance points and project milestones. The task set that best accommodates the needs of the project and characteristics of the team is chosen. The framework described in the generic view of software engineering is completed by a number of umbrella activities. Typical activities in this category include:

- **Software Project Tracking and Control**-allows the software team to assess progress against the project plan and take the necessary action to maintain schedule.
- **Risk Management**-assess the risks that may effect the outcome of the project or the quality of the product.
- **Software Quality Assurance**-defines and conducts the activities required to ensure software quality.
- **Formal Technical Reviews**-assesses software engineering work products in an effort to uncover or remove errors before they are propagated to the next action or activity.
- **Measurement**-defines and collects process, project and product measures that assist the team in delivering software that meets customer needs.
- **Software Configuration Management**-manages the effects of change throughout the software process.
- **Reusability Management**-defines criteria for work product reuse and establishes mechanisms to achieve reusable components.
- **Work Product Preparation and Production**-encompasses the activities required to create work products such as models, documents, logs, forms and lists.

All process models can be categorized within the process framework discussed. But process models do differ fundamentally in:

- The overall flow of activities and tasks and the interdependencies among activities and tasks.
- The degree to which work tasks are defined within each framework activity.
- The degree to which work products are identified and required.
- The manner which quality assurance activities are applied.
- The manner in which project tracking and control activities are applied.
- The overall degree of detail and rigor with which the process is described.
- The degree to which customer and other stakeholders are involved within the project.
- The level of autonomy given to the software project team.

A PROCESS FRAMEWORK

A process framework establishes the foundation for a complete software process by identifying a small number of framework activities that are applicable to all software projects, regardless of their size or complexity. In addition, the process framework encompasses a set of umbrella activities that are applicable across the entire software process.

Each framework activity is populated by a set of software engineering actions-a collection of related tasks that produces a major software engineering work product. Each action is populated with individual work tasks that accomplish some part of the work implied by the

action.

The following generic process framework is applicable to vast majority of software projects:

- **Communication:** this framework activity involves heavy communication and collaboration with the customer and encompasses requirements gathering and other related activities.
- **Planning:** this activity establishes a plan for the software engineering work that follows. It describes the technical tasks to be conducted, the risks that are likely, the resources that will be required, the work products to be produced, and a work schedule.
- **Modeling:** the activity encompasses the creation of models that allow the developer and the customer to better understand software requirements and the design that will achieve those requirements.
- **Construction:** this activity combines code generation and the testing that is required to uncover errors in the code.
- **Deployment:** the software is delivered to the customer who evaluates the delivered product and provides feedback based on the evaluation.

The modeling activity is composed of two software engineering actions:

- **Analysis** encompasses a set of work tasks requirements gathering, elaboration, negotiation, specification and validation that lead to the creation of the analysis model or requirements specification.
- **Design** encompasses work tasks data design, architectural design, interface design and component-level design and create a design model or design specification.

Each software engineering action is represented by a number of different task sets-each a collection of software engineering work tasks, related work products, quality assurance points and project milestones. The task set that best accommodates the needs of the project and characteristics of the team is chosen. The framework described in the generic view of software engineering is completed by a number of umbrella activities. Typical activities in this category include:

- **Software Project Tracking and Control**-allows the software team to assess progress against the project plan and take the necessary action to maintain schedule.
- **Risk Management**-assess the risks that may effect the outcome of the project or the quality of the product.
- **Software Quality Assurance**-defines and conducts the activities required to ensure software quality.
- **Formal Technical Reviews**-assesses software engineering work products in an effort to uncover or remove errors before they are propagated to the next action or activity.
- **Measurement**-defines and collects process, project and product measures that assist the team in delivering software that meets customer needs.
- **Software Configuration Management**-manages the effects of change throughout the software process.
- **Reusability Management**-defines criteria for work product reuse and establishes mechanisms to achieve reusable components.
- **Work Product Preparation and Production**-encompasses the activities required to create work products such as models, documents, logs, forms and lists.

All process models can be categorized within the process framework discussed. But process models do differ fundamentally in:

- The overall flow of activities and tasks and the interdependencies among activities and tasks.
- The degree to which work tasks are defined within each framework activity.

- The degree to which work products are identified and required.
- The manner which quality assurance activities are applied.
- The manner in which project tracking and control activities are applied.
- The overall degree of detail and rigor with which the process is described.
- The degree to which customer and other stakeholders are involved within the project.
- The level of autonomy given to the software project team.

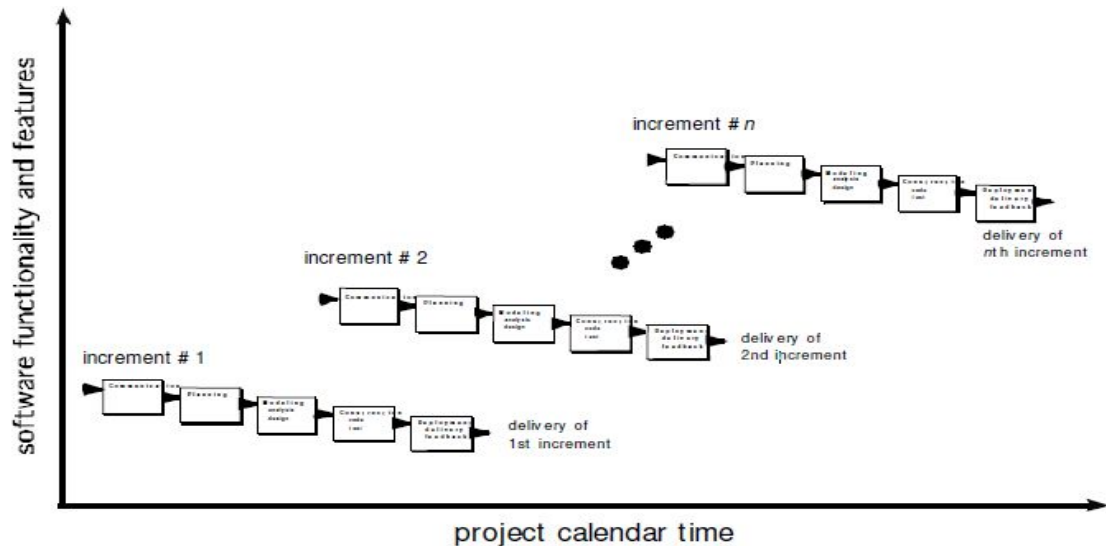
The degree to which team organization and roles are prescribed.

4. Discuss the various life cycle models in software development. (*Apr 11, 16 , Nov 14*)

There are situations in which the initial software requirements are reasonably well-defined. There may be a need to provide the software functionality to users quickly and then refine and expand on that functionality. In such cases a process model that is designed to produce the software in increments is chosen

The Incremental Model

The incremental model combines the elements of the waterfall model applied in an iterative fashion.



THE INCREMENTAL MODEL

The incremental model applies linear sequences in a staggered fashion as calendar time progresses. Each linear sequence produces deliverable increments of the software. When an incremental model is used:

- The first increment is often called the core product.
- The basic requirements are addressed.
- Many supplementary features remain undelivered.
- The core product is used by the customer.
- As a result of use and/or evaluation, a plan is developed for the next increment.
- The plan includes modification of the core product to better meet the needs of the customer and the delivery of additional features and functionality.
- The process is repeated until the complete product is delivered.

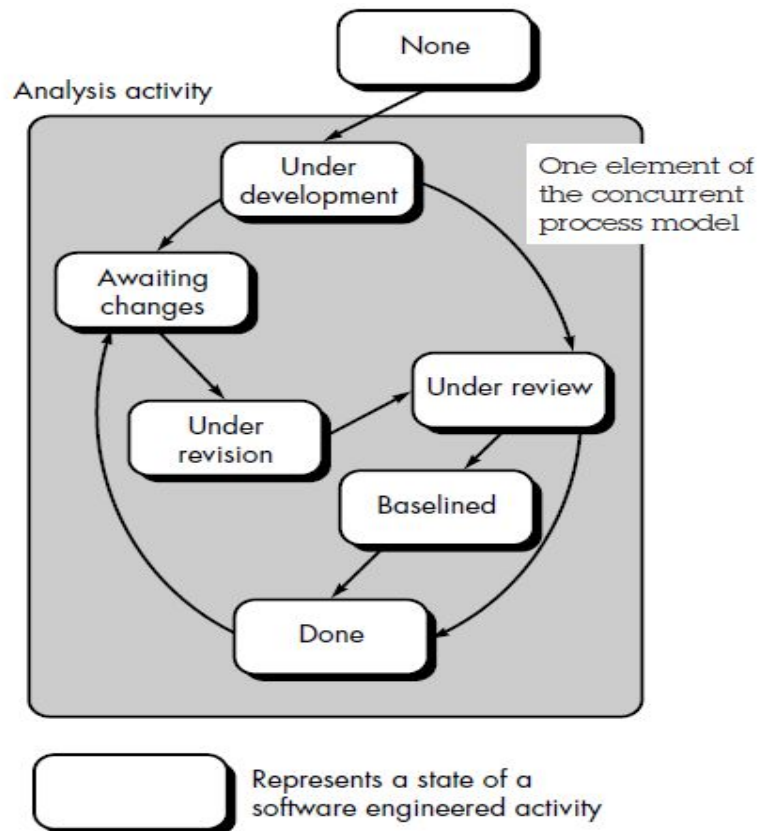
The incremental process model, like prototyping is iterative in nature. But unlike prototyping, the incremental model focuses on the delivery of an operational product with each increment.

- Early increments are stripped down versions of the final product, but they do provide capability that serves the user and platform for evaluation by the user.
- Incremental development is particularly useful when staffing is unavailable for a complete implementation by the business deadline.

- Early increments are implemented with fewer people.

If the core product is received well, additional staff can be added to implement the next increment.

THE CONCURRENT DEVELOPMENT MODEL



The Concurrent Development Model also called Concurrent Engineering, is represented as a series of

- Framework activities
- Software engineering actions and tasks
- And their associated states

The figure provides a representation of one software engineering task within the modeling activity for the concurrent process model. The activity-modeling may be in any one of the states noted at a given time. Similarly other activities or tasks can be represented in an analogous manner. All activities exist concurrently but reside in different states.

For example

- Early in a project the communication activity has completed its first iteration and exists in the awaiting changes state.
- The modeling activity which exists in the none state while initial communication was completed, now makes a transition into the under development state.
- If customer indicates changes in the requirements, the modeling activity moves from the under development state into the awaiting changes state.

The concurrent process model defines a series of events that will trigger transitions from state to state for each of the software engineering activities, actions or tasks. The concurrent process model is applicable to all types of software development and provides an accurate picture of the current state of a project. This model defines a network of activities. Events triggered at one point in the process network trigger transitions among the states.

5. Discuss about the COCOMO Models(Basic, Intermediate and Detailed)for cost estimation. (Apr 15)

Barry Boehm introduced COCOMO II (COSt CONstructive MOdel) which is an hierarchy of estimation models that addresses the following areas:

- Application composition model. Used during the early stages of software engineering, when prototyping of user interfaces, consideration of software and system interaction, assessment of performance, and evaluation of technology maturity are paramount.
- 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.

Three different sizing options are available as part of the model hierarchy: object points, function points, and lines of source code.

OBJECT TYPE	COMPLEXITY WEIGHT		
	SIMPLE	MEDIUM	DIFFICULT
Screen	1	2	3
Report	2	5	8
3GL component	4	3	10

The object point is an indirect software measure that is computed using counts of the number of (1) screens (at the user interface), (2) reports, and (3) components likely to be required to build the application.

When component-based development or general software reuse is to be applied, the percent of reuse (%reuse) is estimated and the object point count is adjusted:

$$NOP = (\text{Object points}) * [(100 - \% \text{reuse}) / 100]$$

where NOP is defined as new object points.

To derive an estimate of effort based on the computed NOP value, a “productivity rate” must be derived.

$$\text{PROD} = \frac{\text{NOP}}{\text{Person-month}}$$

for different levels of developer experience and development environment maturity.

Once the productivity rate has been determined, an estimate of project effort is computed using

$$\text{Estimated effort} = \frac{\text{NOP}}{\text{PROD}}$$

In more advanced COCOMO II models, 12 a variety of scale factors, cost drivers, and adjustment procedures are required.

The Software Equation

The *software equation* is a dynamic multivariable model that assumes a specific distribution of effort over the life of a software development project.

$$E = \frac{\text{LOC} * B^{0.333}}{P^3} * \frac{1}{t^4}$$

where

E --effort in person-months or person-years

t --project duration in months or years

B-- "special skills factor"13

P -- "productivity parameter" that reflects: overall process maturity and management practices, the extent to which good software engineering practices are used, the level of

programming languages used, the state of the software environment, the skills and experience of the software team, and the

complexity of the application software equation has two independent parameters:

- an estimate of size (in LOC) and
- an indication of project duration in calendar months or years.

To simplify the estimation process and use a more common form for their estimation model, Putnam and Myers suggest a set of equations derived from the software equation. Minimum development time is defined as

LOC					
Developer's experience/capability	Very low	Low	Nominal	High	Very high
Environment maturity/capability	Very low	Low	Nominal	High	Very high
PROD	4	7	13	25	50

$$T_{min} = 8.14 \frac{P^{0.43}}{LOC} \text{ IN months for } t_{min} > 6 \text{ months}$$

$$E = 180 B t^3 \text{ in person-months for } E \geq 20 \text{ person-months}$$

6. Describe at least one scenario where

- RAD model would be applicable and not waterfall model.
- Waterfall model is preferable to all other models.

RAD Model would be applicable and not waterfall model

The RAD model is suitable for information system applications, business applications and for systems that can be modularized because of the following reasons:

- This model is similar to waterfall model but uses very short development cycle.
- It uses component based construction and emphasizes reuse and code generation.
- This model uses multiple teams on scalable projects.
- The RAD model is suitable for the projects where technical risks are not high.
- The RAD model requires heavy resources.

In waterfall model "blocking state" situation occurs. In blocking state situation, project team members have to wait for other members of the team to complete the dependent tasks. For high speed and short time development projects, RAD model would be applicable, where as waterfall model is unsuited for these types of projects.

Waterfall model is preferable to all other models

- Waterfall model is useful for the projects in which the requirements are well understood and unlikely to change radically during the system development.
- The waterfall model is simple to implement, compared to all other software models.
- For implementation of small systems waterfall model is useful.
- Waterfall model suggests a systematic, sequential approach to software development.
- This model consists of information domain, function, and behavioral requirements of the system.

It is the oldest software paradigm. It is also called as "linear-sequential model" or "classic lifecycle model".

7. Compare and contrast the different lifecycle models. (Nov 11)

PROCESS MODEL	CONCEPT	ADVANTAGE	DISADVANTAGE
The waterfall model	The waterfall model, called the classic life cycle suggests a systematic, sequential approach to software development.	It provides a template into which methods for analysis, design, and other phases can be placed. It provides for baseline management. It is better than any haphazard approach to software development.	It lacks the perception for a reverse engineering on how to engineer an existing legacy system. The client has to wait until the installation and checkout phase to see how a system works. Thus a complex system requires considerable time and effort. Real time software cannot follow this model. Customer satisfaction is not full filled.
The incremental model	The incremental model combines the elements of the waterfall model applied in an iterative fashion. The incremental model applies linear sequences in a staggered fashion as calendar time progresses. Each linear sequence produces deliverable increments of the software.	The software development activities are repeated each time there is a new release of software. It provides a platform for evaluation by the user. It can be planned to manage technical risks. It enables partial functionality to be delivered to end users without ordinary delay.	It makes the Unrealistic assumptions that system as well as Software requirements remain stable which is not true.
Prototyping	The prototyping paradigm assists the software engineer and the customer to better understand what is to be built when the	It produces the products quickly and thus saves the time. It solves the waiting problem in waterfall model. It	It ignores quality, reliability maintainability and safety requirements. Customer satisfaction is not achieved

		requirements are fuzzy. The customer usually defines a set of general objectives for software, but does not identify detailed input, processing or output requirements.	minimizes the cost and product failure. It is possible for the developers and client to check the function of preliminary implementations of system models before committing to a final system. It obtains feedback from clients and changes in system concept.	
	Spiral model	The spiral model is an evolutionary software process model that couples the iterative nature of prototyping with the controlled and systematic aspects of the waterfall model. It provides the potential for rapid development of increasingly more complex versions of the software.	It provides the potential for rapid development for incremental versions of the software. It can be applied throughout the life of the computer software. It allows the developer to apply the prototyping approach at any stage. It demands a direct consideration of technical risks at all stages.	It may be difficult to convince the customers at times especially in contrast solutions. It demands considerable risks assessment expertise and relies on them for success. It takes time for determining the efficacy and thus the model cannot be used as widely as others.
	RAD model	Rapid Application Development (RAD) is an incremental software process model that emphasizes a short development cycle.	If requirements are well understood and project scope is constrained, the RAD process enables a development team	RAD requires sufficient human resources to create the right number of RAD teams. If developers and customers are not committed RAD

		The RAD model is a high-speed adaptation of the waterfall model, in which rapid development is achieved by using component-based construction approach.	to create a fully functional system within a very short time period.	project fails. If a system is not properly modularized, building the components for RAD will be problematic. High performance cannot be achieved. Not appropriate when technical risks are high or when new technology is used.
8.	<p>Discuss briefly about the techniques used for Project scheduling.</p> <p>Software project scheduling is an action that distributes estimated effort across the planned project duration by allocating the effort to specific software engineering tasks. Scheduling for software engineering projects can be viewed from two rather different perspectives. In the first, an end date for release of a computer-based system has already (and irrevocably) been established. The software organization is constrained to distribute effort within the prescribed time frame. The second view of software scheduling assumes that rough chronological bounds have been discussed but that the end date is set by the software engineering organization.</p> <p>Basic Principles</p> <ul style="list-style-type: none"> • Compartmentalization. The project must be compartmentalized into a number of manageable activities and tasks. To accomplish compartmentalization, both the product and the process are refined. • Interdependency. The interdependency of each compartmentalized activity or task must be determined. Some tasks must occur in sequence, while others can occur in parallel. Some activities cannot commence until the work product produced by another is available. Other activities can occur independently. • Time allocation. Each task to be scheduled must be allocated some number of work units (e.g., person-days of effort). In addition, each task must be assigned a start date and a completion date that are a function of the interdependencies and whether work will be conducted on a full-time or part-time basis. • Effort validation. Every project has a defined number of people on the software team. As time allocation occurs, you must ensure that no more than the allocated number of people has been scheduled at any given time. For example, consider a project that has three assigned software engineers (e.g., three person-days are available per day of assigned effort). On a given day, seven concurrent tasks must be accomplished. Each task requires 0.50 person-days of effort. More effort has been allocated than there are people to do the work. • Defined responsibilities. Every task that is scheduled should be assigned to a specific team member. • Defined outcomes. Every task that is scheduled should have a defined outcome. For software projects, the outcome is normally a work product (e.g., the design of a component) or a part of a work product. Work products are often combined in deliverables. • Defined milestones. Every task or group of tasks should be associated with a project milestone. A milestone is accomplished when one or more work products has been reviewed for quality. 			

The Relationship Between People and Effort

L , is related to effort and development time by the equation:

$$L = P * E^{1/3} t^{4/3}$$

Rearranging this software equation, we can arrive at an expression for development effort E :

$$E = \frac{L^3}{P^3 t^4}$$

Effort Distribution

A recommended distribution of effort across the software process is often referred to as the 40–20–40 rule. Forty percent of all effort is allocated to frontend analysis and design. A similar percentage is applied to back-end testing. 20 percent of effort is deemphasized in the coding. Because of the effort applied to software design, code should follow with relatively little difficulty. A range of 15 to 20 percent of overall effort can be achieved. Testing and subsequent debugging can account for 30 to 40 percent of software development effort. The criticality of the software often dictates the amount of testing that is required. If software is human rated (i.e., software failure can result in loss of life), even higher percentages are typical.

9. Discuss LOC and FP- Cost Estimation Models in detail.

Size-Oriented Metrics

- LOC measure claim that LOC is an “artifact” of all software development projects that can be easily counted, that many existing software estimation models use LOC or KLOC as a key input, and that a large body of literature and data predicated on LOC already exists.
- The planner must estimate the LOC to be produced long before analysis and design has been completed.

Function-Oriented Metrics

- Function-oriented software metrics use a measure of the functionality delivered by the application as a normalization value. The most widely used function-oriented metric is the function point (FP). Computation of the function point is based on characteristics of the software’s information domain and complexity.
- The function point, like the LOC measure, is controversial. Proponents claim that FP is programming language independent, making it ideal for applications using conventional and nonprocedural languages, and that it is based on data that are more likely to be known early in the evolution of a project, making FP more attractive as an estimation approach.
- Opponents claim that the method requires some “sleight of hand” in that computation is based on subjective rather than objective data, that counts of the information domain (and other dimensions) can be difficult to collect after the fact, and that FP has no direct physical meaning – it’s just a number.

Reconciling LOC and FP Metrics

The relationship between lines of code and function points depends upon the programming language that is used to implement the software and the quality of the design. Function points and LOC-based metrics have been found to be relatively accurate predictors of software development effort and cost. However, in order to use LOC and FP for estimation, an historical baseline of information must be established.

Within the context of process and project metrics, productivity and quality should be concerned which are the measures of software development “output” as a function of effort and time applied and measures of the “fitness for use” of the work products that are produced. For

	process improvement and project planning purposes, the interest is historical.
10	<p>Give a brief account about Risk Management. (<i>Nov 16</i>)</p> <p>Project risks threaten the project plan. That is, if project risks become real, it is likely that the project schedule will slip and that costs will increase. Project risks identify potential budgetary, schedule, personnel (staffing and organization), resource, stakeholder, and requirements problems and their impact on a software project.</p> <p>Technical risks threaten the quality and timeliness of the software to be produced. If a technical risk becomes a reality, implementation may become difficult or impossible. Technical risks identify potential design, implementation, interface, verification, and maintenance problems.</p> <p>Business risks threaten the viability of the software to be built and often jeopardize the project or the product. Candidates for the top five business risks are</p> <ul style="list-style-type: none"> • building an excellent product or system that no one really wants (market risk), • building a product that no longer fits into the overall business strategy for the company (strategic risk), • building a product that the sales force doesn't understand how to sell (sales risk), • losing the support of senior management due to a change in focus or a change in people (management risk), and • losing budgetary or personnel commitment (budget risks). <p>Known risks are those that can be uncovered after careful evaluation of the project plan, the business and technical environment in which the project is being developed, and other reliable information sources (e.g., unrealistic delivery date, lack of documented requirements or software scope, poor development environment).</p> <p>Predictable risks are extrapolated from past project experience (e.g., staff turnover, poor communication with the customer, dilution of staff effort as on-going maintenance requests are serviced). Unpredictable risks are the joker in the deck. They can and do occur, but they are extremely difficult to identify in advance.</p> <p>Risk Identification:</p> <ul style="list-style-type: none"> • Risks are about events that, when triggered, cause problems or benefits. Hence, risk identification can start with the source of our problems and those of our competitors (benefit), or with the problem itself. • Source analysis - Risk sources may be internal or external to the system that is the target of risk management (use mitigation instead of management since by its own definition risk deals with factors of decision-making that cannot be managed). Examples of risk sources are: stakeholders of a project, employees of a company or the weather over an airport. • Problem analysis- Risks are related to identified threats. For example: the threat of losing money, the threat of abuse of confidential information or the threat of human errors, accidents and casualties. The threats may exist with various entities, most important with shareholders, customers and legislative bodies such as the government. <p>Risk Assessment:</p> <p>Once risks have been identified, they must then be assessed as to their potential severity of impact (generally a negative impact, such as damage or loss) and to the probability of occurrence. These quantities can be either simple to measure, in the case of the value of a lost building, or impossible to know for sure in the case of the probability of an unlikely event occurring. Therefore, in the assessment process it is critical to make the best educated decisions in order to properly prioritize the implementation of the <u>risk management plan</u>.</p>

	<p>Composite Risk Index = Impact of Risk event x Probability of Occurrence</p> <p>Risk mitigation:</p> <p>Risk mitigation measures are usually formulated according to one or more of the following major risk options, which are:</p> <ul style="list-style-type: none"> • Design a new business process with adequate built-in risk control and containment measures from the start. • Periodically re-assess risks that are accepted in ongoing processes as a normal feature of business operations and modify mitigation measures. • Transfer risks to an external agency (e.g. an insurance company) • Avoid risks altogether (e.g. by closing down a particular high-risk business area) <p>Prioritizing the risk management processes too highly could keep an organization from ever completing a project or even getting started. This is especially true if other work is suspended until the risk management process is considered complete.</p>
11	<p>What is CMMI? Explain the different levels of CMMI and its goals?(Nov 17, 18)</p> <p>The CMMI represents a process meta-model in two different ways: (1) as a “continuous” model and (2) as a “staged” model. Each process area (e.g., project planning or requirements management) is formally assessed against specific goals and practices and is rated according to the following capability levels:</p> <p>Level 0: Incomplete—the process area (e.g., requirements management) is either not performed or does not achieve all goals and objectives defined by the CMMI for level 1 capability for the process area.</p> <p>Level 1: Performed—all of the specific goals of the process area (as defined by the CMMI) have -been satisfied. Work tasks required to produce defined work products are being conducted.</p> <p>Level 2: Managed—all capability level 1 criteria have been satisfied. In addition, all work associated with the process area conforms to an organizationally defined policy; all people doing the work have access to adequate resources to get the job done; stakeholders are actively involved in the process area as required; all work tasks and work products are “monitored, controlled, and reviewed; and are evaluated for adherence to the process description” .</p> <p>Level 3: Defined—all capability level 2 criteria have been achieved. In addition, the process is “tailored from the organization’s set of standard processes according to the organization’s tailoring guidelines, and contributes work products, measures, and other process-improvement information to the organizational process assets” .</p> <p>Level 4: Quantitatively managed—all capability level 3 criteria have been achieved. In addition, the process area is controlled and improved using measurement and quantitative assessment. “Quantitative objectives for quality and process performance are established and used as criteria in managing the process” .</p> <p>Level 5: Optimized—all capability level 4 criteria have been achieved. In addition, the process area is adapted and optimized using quantitative (statistical) means to meet changing customer needs and to continually improve the efficacy of the process area under consideration.</p> <p>The CMMI defines each process area in terms of “specific goals” and the “specific practices” required to achieve these goals. <i>Specific goals</i> establish the characteristics that must exist if the activities implied by a process area are to be effective. <i>Specific practices</i> refine a goal into a set of process-related activities.</p>
12	<p>Explain Specialized Process and Unified Process Models in detail.</p> <p>The component-based development model incorporates many of the characteristics of the spiral</p>

model. It is evolutionary in nature, demanding an iterative approach to the creation of software. However, the component-based development model constructs applications from pre-packaged software components.

The component-based development model incorporates the following steps:

- Available component-based products are researched and evaluated for the application domain in question.
- Component integration issues are considered.
- A software architecture is designed to accommodate the components.
- Components are integrated into the architecture.
- Comprehensive testing is conducted to ensure proper functionality.

Formal Methods Model

The formal methods model encompasses a set of activities that leads to formal mathematical specification of computer software.

- The development of formal models is currently quite time consuming and expensive.
- Because few software developers have the necessary background to apply formal methods, extensive training is required.
- It is difficult to use the models as a communication mechanism for technically unsophisticated customers.

Aspect-oriented software development

Aspect-oriented software development (AOSD), often referred to as aspect-oriented programming (AOP), is a relatively new software engineering paradigm that provides a process and methodological approach for defining, specifying, designing, and constructing aspects—“mechanisms beyond subroutines and inheritance for localizing the expression of a crosscutting concern”.

Unified Process Model

The Unified Process recognizes the importance of customer communication and streamlined methods for describing the customer’s view of a system. It emphasizes the important role of software architecture and “helps the architect focus on the right goals, such as under standability, reliance to future changes, and reuse” [Jac99]. It suggests a process flow that is iterative and incremental, providing the evolutionary feel that is essential in modern software development

Phases of Unified Process Model

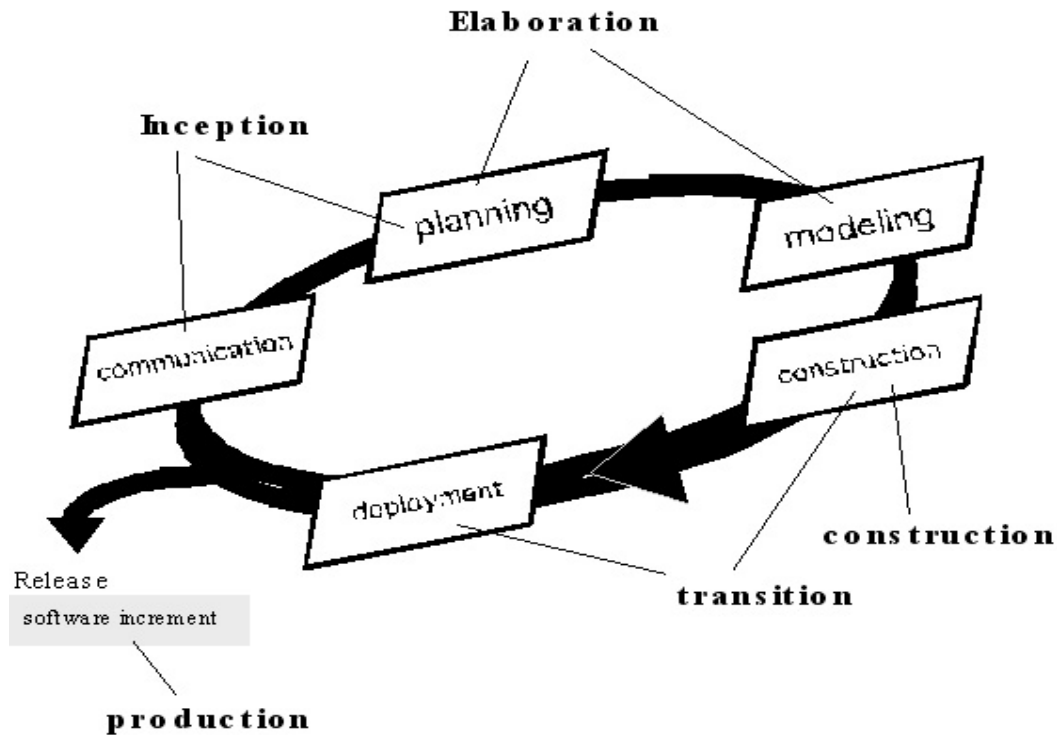
The **inception phase** of the UP encompasses both customer communication and planning activities. By collaborating with stakeholders, business requirements for the software are identified; a rough architecture for the system is proposed; and a plan for the iterative, incremental nature of the ensuing project is developed. Fundamental business requirements are described through a set of preliminary use cases that describe which features and functions each major class of users desires.

The **elaboration phase** encompasses the communication and modelling activities of the generic process model. Elaboration refines and expands the preliminary use cases that were developed as part of the inception phase and expands the architectural representation to include five different views of the software—the use case model, the requirements model, the design model, the implementation model, and the deployment model.

The **construction phase** of the UP is identical to the construction activity defined for the generic software process. Using the architectural model as input, the construction phase develops or acquires the software components that will make each use case operational for end users.

The **transition phase** of the UP encompasses the latter stages of the generic construction activity and the first part of the generic deployment (delivery and feedback) activity. Software is given

to end users for beta testing and user feedback reports both defects and necessary changes. The **production phase** of the UP coincides with the deployment activity of the generic process. During this phase, the on-going use of the software is monitored, support for the operating environment (infrastructure) is provided, and defect reports and requests for changes are submitted and evaluated.



- 13 An application has the following :10 low external Inputs ,12 High External Outputs,20 low Internal Logical Files ,15 High External interface Files ,12 Average External Inquiries and a value adjustment factor of 1.10. What is the unadjusted and adjusted function point count? (Apr 17)

$$FP = \text{count total} * [0.65 + 0.01 * \sum f_i]$$

Adjustment factor given, $\sum f_i = 1.10$
 So, Adjusted function points = $452 * [1.75]$

=791 adjusted function points.

Measurement Parameter	count	Weighting Factor			Total
		Simple	Average	Complex	
External inputs	10	3	4	6	30
External outputs	12	4	5	7	84
internal logical files	20	7	10	15	140
external interface files	15	5	7	10	150
External inquiries	12	3	4	6	48
					452

- 14 How function point analysis methodology is applied in estimation of software of software size? Explain .Why FPA methodology is better than LOC methodology?(Apr 17)

Decomposition for FP-based estimation focuses on information domain values rather than software functions, you would estimate inputs, outputs, inquiries, files, and external interfaces for the CAD software. For the purposes of this estimate, the complexity weighting factor is assumed to be average.

Information domain value	Opt.	Likely	Pess.	Est. count	Weight	FP count
Number of external inputs	20	24	30	24	4	97
Number of external outputs	12	15	22	16	5	78
Number of external inquiries	16	22	28	22	5	88
Number of internal logical files	4	4	5	4	10	42
Number of external interface files	2	2	3	2	7	15
Count total						320

Each of the complexity weighting factors is estimated, and the value adjustment factor is computed as below.

Factor	Value
Backup and recovery	4
Data communications	2
Distributed processing	0
Performance critical	4
Existing operating environment	3
Online data entry	4
Input transaction over multiple screens	5
Master files updated online	3
Information domain values complex	5
Internal processing complex	5
Code designed for reuse	4
Conversion/installation in design	3
Multiple installations	5
Application designed for change	5
Value adjustment factor	1.17

Finally, the estimated number of FP is derived: $FP \text{ estimated count total} \times [0.65 + 0.01 \times (F_i)]$ 375

The organizational average productivity for systems of this type is 6.5 FP/pm. Based on a burdened labor rate of \$8000 per month, the cost per FP is approximately \$1230. Based on the FP estimate and the historical productivity data, the total estimated project cost is \$461,000 and the estimated effort is 58 person-months.

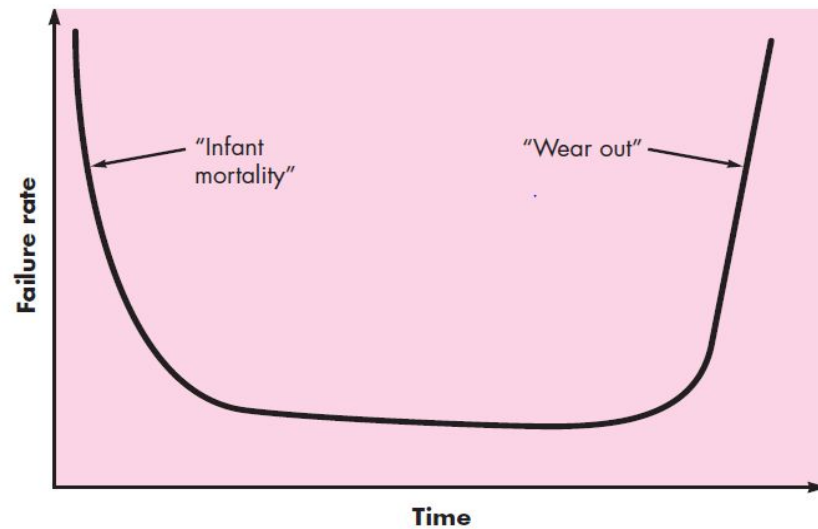
The LOC and FP estimation techniques differ in the level of detail required for decomposition and the target of the partitioning. When LOC is used as the estimation variable, decomposition is absolutely essential and is often taken to considerable levels of detail. The greater the degree of partitioning, the more likely reasonably accurate estimates of LOC can be developed.

For FP estimates, decomposition works differently. Rather than focusing on function, each of the information domain characteristics—inputs, outputs, data files, inquiries, and external interfaces—as well as the 14 complexity adjustment values discussed in Chapter 23 are estimated. The resultant estimates can then be used to derive an FP value that can be tied to past data and used to generate an estimate.

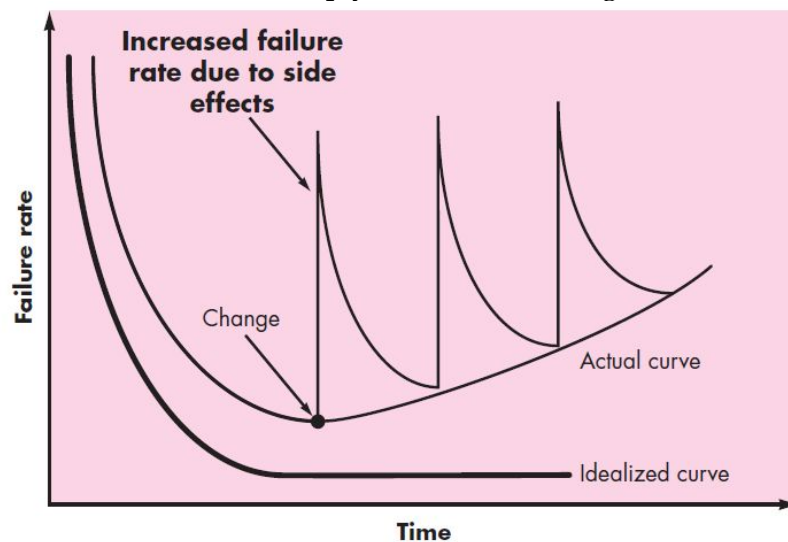
- 15 i)What is the impact of reusability in software development process?(Nov 17)

Reuse is the use of work products (such as a code, design and test), which are the products or by-products of the software-development process, without modification in the development of other software . It includes multiple reuse programs in different division within the same company so that it has been largely positive. It maintains the result into product, having higher

	<p>quality, because the work products can be used multiple times. Also, this reusability increases the productivity because it does not require much works for consumers. However, this productivity fails to satisfy the requirements for time-to-market. One of the potential ways for reducing time-to-market is that we efficiently apply the reuse on the critical path of a development project, the chain of activities that determine the total project duration. Reuse allows an organization to use personnel more effectively because it leverages expertise. Leveraged reuse is to modify existing work products to meet specific system requirements. Software experts, who have a lot of experiences, can concentrate on creating work product that can be reused by novice personnel. However, software reuse is not an inexpensive concept because it requires creating and maintaining reusable work products, a reuse library, and reuse tools. In this section, we use an economic analysis method to help evaluate the costs and benefits of reuse.</p> <p>ii) Explain the component based software development model with a neat sketch. (Nov/Dec 17)</p> <p>The component-based development model incorporates many of the characteristics of the spiral model. It is evolutionary in nature, demanding an iterative approach to the creation of software. However, the component-based development model constructs applications from pre-packaged software components.</p> <p>The component-based development model incorporates the following steps:</p> <ul style="list-style-type: none"> • Available component-based products are researched and evaluated for the application domain in question. • Component integration issues are considered. • A software architecture is designed to accommodate the components. • Components are integrated into the architecture. • Comprehensive testing is conducted to ensure proper functionality. <p>Formal Methods Model</p> <p>The formal methods model encompasses a set of activities that leads to formal mathematical specification of computer software.</p> <ul style="list-style-type: none"> • The development of formal models is currently quite time consuming and expensive. • Because few software developers have the necessary background to apply formal methods, extensive training is required. • It is difficult to use the models as a communication mechanism for technically +unsophisticated customers. <p>Aspect-oriented software development</p> <p>Aspect-oriented software development (AOSD), often referred to as aspect-oriented programming (AOP), is a relatively new software engineering paradigm that provides a process and methodological approach for defining, specifying, designing, and constructing aspects—“mechanisms beyond subroutines and inheritance for localizing the expression of a crosscutting concern”.</p>
16	<p>Write a note on the unique characters of software. (Nov 17)</p> <p>CHARACTERISTICS OF SOFTWARE</p> <ol style="list-style-type: none"> 1. Software is developed or engineered; it is not manufactured in the classical sense. <ul style="list-style-type: none"> • Software costs are concentrated in engineering. This means that software projects cannot be managed as if they were manufacturing projects. <ol style="list-style-type: none"> 2. Software doesn't “wear out.”



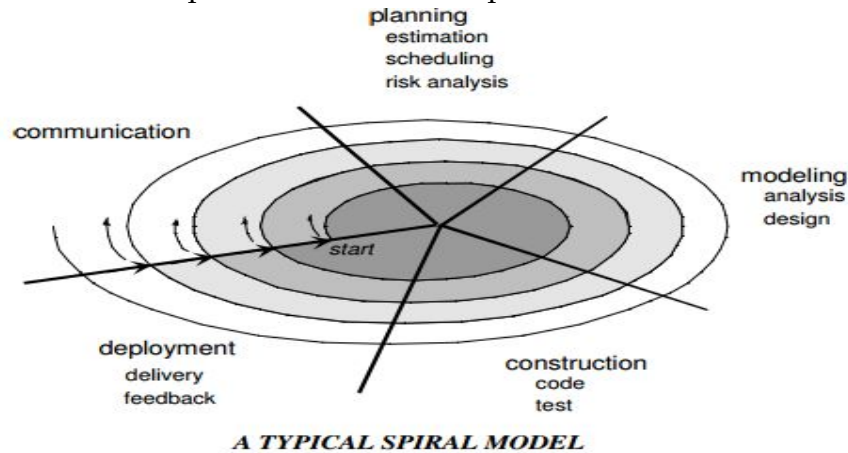
- Figure depicts failure rate as a function of time for hardware. The relationship, often called the “bathtub curve,” indicates that hardware exhibits relatively high failure rates early in its life (these failures are often attributable to design or manufacturing defects); defects are corrected and the failure rate drops to a steady-state level (hopefully, quite low) for some period of time. As time passes, however, the failure rate rises again as hardware components suffer from the cumulative effects of dust, vibration, abuse, temperature extremes, and many other environmental maladies. Stated simply, the hardware begins to wear out.



- Considering the time curve, software will undergo change. As changes are made, it is likely that errors will be introduced, causing the failure rate curve to spike as shown in the “actual curve” (Figure 1.2). Before the curve can return to the original steady-state failure rate, another change is requested, causing the curve to spike again. Slowly, the minimum failure rate level begins to rise – the software is deteriorating due to change.

Although the industry is moving toward component-based construction, most software continues to be custom built. A software component should be designed and implemented so that it can be reused in many different programs. Modern reusable components encapsulate both data and the processing that is applied to the data, enabling the software engineer to create new applications from reusable parts.

17 What is the significance of the spiral model when compared with other models.



A spiral model is divided into a set of framework activities defined by the software engineering team. Each of the framework activities represents one segment of the spiral path as shown in the figure.

- As evolution begins the software team performs activities implied by the by the circuit around the spiral, in clockwise direction, beginning at the center.
- Risk is considered at each revolution made.
- Anchor point milestones-a combination of work products and conditions that are attained along the path of the spiral are noted for each evolutionary pass.
- First circuit around the spiral results in the development of product specifications.
- Subsequent passes develop a prototype, progressively lead to sophisticated version of the software.
- Cost and schedule are adjusted based on feedback derived from the customer after delivery.
- Unlike other process models that end when the software is delivered, the spiral model can be adapted throughout the life of the software.
- The first spiral represents a Concept Development Project which starts at core and continues for multiple iterations until concept development is implemented.
- The concept developed to an actual product, proceeds outward on the spiral and a New Product Development Project commences.

The new product may evolve to represent Product Enhancement Project.

18 List out the various umbrella activities which support software development process and discuss about their necessity in maintaining the quality in both software process and product that is being developed for railway reservation system. (Apr 21)

Umbrella Activities

- software project management
- formal technical reviews
- software quality assurance
- software configuration management
- reusability management

- measurement
- document preparation and production
- risk management

Software Project tracking and Control

Information about the route, cancellation of tickets, departure time, arrival time, number of trains available and other such information are provided. Store and retrieve information about the various transactions related to Rail travel. Keep track of all its passengers and thus schedule their journey accordingly. Maintains records of passengers travelling in the different trains on different dates reaching different destinations in the system. User can enquire about the PNR status, seat availability and trains on a route. User friendly interface to administrator and customer.

Risk Management

Tasks required to assess both technical and management risks.

Software Quality Assurance

Technical Review

Factor	Value
Backup and recovery	5
Data communications	5
Distributed processing	3
Performance critical	4
Existing operating environment	4

On-line data entry	5
Input transaction over multiple screens	4
Master files updated on-line	4
Information domain values complex	5
Internal processing complex	5
Code designed for reuse	4
Conversion/installation in design	4
Multiple installations	5
Application designed for change	5
Total	62

Risk	Description
Project Risks	Identifies potential budgetary, schedule, personnel (staffing and organization), resource, customer, and requirements problems and their impact on a software project. It threatens the project plan. That is, if project risks become real, it is likely that project schedule will slip and that costs will increase.
Technical Risks	Identifies potential design, implementation, interface, verification, and maintenance problems. Technical risks threaten the quality and timeliness of the software to be produced. If a technical risk becomes a reality, implementation may become difficult or impossible.

Each of these umbrella activities is defined by a set of tasks that are adapted to the project type and degree of rigor with which software engineering is to be applied.

- 19 Explain how work break down structure is used in software engineering? Discuss how software project scheduling helps in timely release of a product? (*Apr 18*)

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

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

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

Some use tree structure to illustrate the WBS, while others use lists and tables. Outlining is one of the easiest ways of representing a WBS.

Following example is an outlined WBS:

Project Name			
	Task 1		
		Subtask 1.1	Work Package 1.1.1 Work Package 1.1.2
		Subtask 1.2	Workpackage 1.2.1 Workpackage 1.2.2
	Task 2		
		Subtask 2.1	Workpackage 2.1.1 Workpackage 2.1.2

There are many design goals for WBS. Some important goals are as follows:

- Giving visibility to important work efforts.
- Giving visibility to risky work efforts.
- Illustrate the correlation between the activities and deliverables.
- Show clear ownership by task leaders.

WBS Diagram

In a WBS diagram, the project scope is graphically expressed. Usually the diagram starts with a graphic object or a box at the top, which represents the entire project. Then, there are sub-components under the box.

These boxes represent the deliverables of the project. Under each deliverable, there are sub-elements listed. These sub-elements are the activities that should be performed in order to achieve the deliverables.

Although most of the WBS diagrams are designed based on the deliveries, some WBS are created based on the project phases. Usually, information technology projects are perfectly fit

into WBS model.

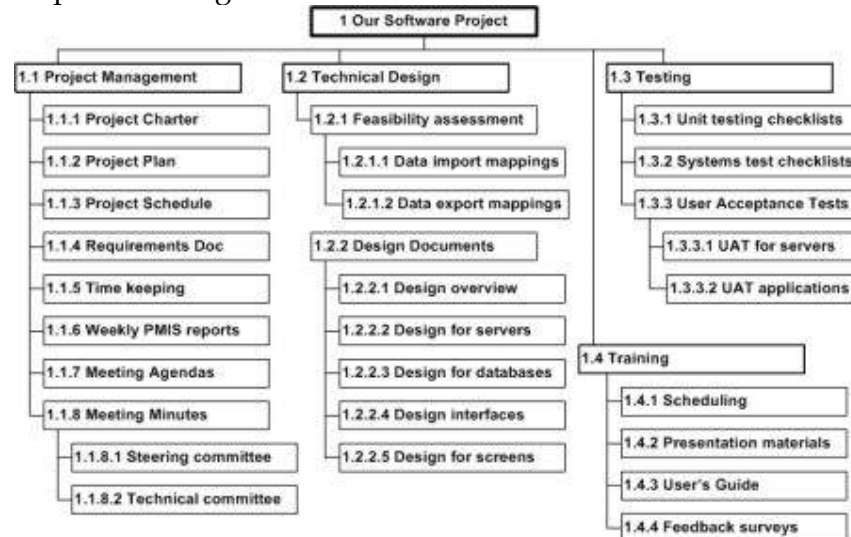
Therefore, almost all information technology projects make use of WBS.

In addition to the general use of WBS, there is specific objective for deriving a WBS as well.

WBS is the input for Gantt charts, a tool that is used for project management purpose.

Gantt chart is used for tracking the progression of the tasks derived by WBS.

Following is a sample WBS diagram:

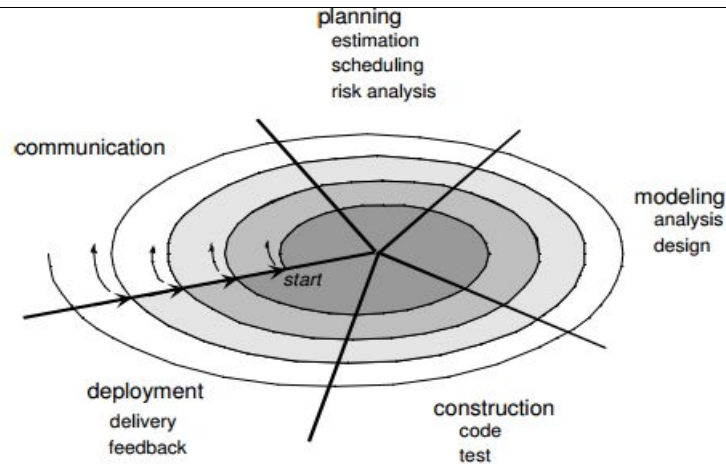


Software project scheduling is an action that distributes estimated effort across the planned project duration by allocating the effort to specific software engineering tasks. Scheduling for software engineering projects can be viewed from two rather different perspectives. In the first, an end date for release of a computer-based system has already (and irrevocably) been established. The software organization is constrained to distribute effort within the prescribed time frame. The second view of software scheduling assumes that rough chronological bounds have been discussed but that the end date is set by the software engineering organization.

Basic Principles

- **Compartmentalization.** The project must be compartmentalized into a number of manageable activities and tasks. To accomplish compartmentalization, both the product and the process are refined.
- **Interdependency.** The interdependency of each compartmentalized activity or task must be determined. Some tasks must occur in sequence, while others can occur in parallel. Some activities cannot commence until the work product produced by another is available. Other activities can occur independently.
- **Time allocation.** Each task to be scheduled must be allocated some number of work units (e.g., person-days of effort). In addition, each task must be assigned a start date and a completion date that are a function of the interdependencies and whether work will be conducted on a full-time or part-time basis.
- **Effort validation.** Every project has a defined number of people on the software team. As time allocation occurs, you must ensure that no more than the allocated number of people has been scheduled at any given time. For example, consider a project that has three assigned software engineers (e.g., three person-days are available per day of assigned effort). On a given day, seven concurrent tasks must be accomplished. Each task requires 0.50 person-days of effort. More effort has been allocated than there are people to do the work.
- **Defined responsibilities.** Every task that is scheduled should be assigned to a specific team member.

	<ul style="list-style-type: none"> • Defined outcomes. Every task that is scheduled should have a defined outcome. For software projects, the outcome is normally a work product (e.g., the design of a component) or a part of a work product. Work products are often combined in deliverables. • Defined milestones. Every task or group of tasks should be associated with a project milestone. A milestone is accomplished when one or more work products has been reviewed for quality. <p>The Relationship Between People and Effort</p> <p>L, is related to effort and development time by the equation:</p> $L = P \cdot E^{1/3} t^{4/3}$ <p>Rearranging this software equation, we can arrive at an expression for development effort E:</p> $E = \frac{L^3}{P^3 t^4}$ <p>Effort Distribution</p> <p>A recommended distribution of effort across the software process is often referred to as the 40–20–40 rule. Forty percent of all effort is allocated to frontend analysis and design. A similar percentage is applied to back-end testing. 20 percent of effort is deemphasized in the coding. Because of the effort applied to software design, code should follow with relatively little difficulty. A range of 15 to 20 percent of overall effort can be achieved. Testing and subsequent debugging can account for 30 to 40 percent of software development effort. The criticality of the software often dictates the amount of testing that is required. If software is human rated (i.e., software failure can result in loss of life), even higher percentages are typical.</p>
20	<p>Which software process model is good for risk management? Explain the model. Describe how the model is used to layout the objectives, risks and plans for quality improvement. (<i>Apr 18</i>)</p> <p>SPIRAL MODEL</p> <p>The spiral model is an evolutionary software process model that couples the iterative nature of prototyping with the controlled and systematic aspects of the waterfall model. It provides the potential for rapid development of increasingly more complex versions of the software. Boehm definition: The spiral development model is a risk-driven process model generator. It has two main distinguishing features :</p> <ul style="list-style-type: none"> • One is a cyclic approach for incrementally growing a system's degree of definition and implementation while decreasing its degree of risk. • The other is a set of anchor point milestones for ensuring stakeholder commitment to feasible and mutually satisfactory system solutions. <p>Using spiral model, software is developed in a series of evolutionary releases. During early iterations, the release is a paper model or prototype. During later iterations, more complex versions of the engineered system are produced.</p>



A TYPICAL SPIRAL MODEL

A spiral model is divided into a set of framework activities defined by the software engineering team. Each of the framework activities represents one segment of the spiral path as shown in the figure.

- • As evolution begins the software team performs activities implied by the by the circuit around the spiral, in clockwise direction, beginning at the center.
- • Risk is considered at each revolution made.
- • Anchor point milestones-a combination of work products and conditions that are attained along the path of the spiral are noted for each evolutionary pass.
- • First circuit around the spiral results in the development of product specifications.
- • Subsequent passes develop a prototype, progressively lead to sophisticated version of the software.
- • Cost and schedule are adjusted based on feedback derived from the customer after delivery.
- • Unlike other process models that end when the software is delivered, the spiral model can be adapted throughout the life of the software.
- • The first spiral represents a Concept Development Project which starts at core and continues for multiple iterations until concept development is implemented.
- • The concept developed to an actual product, proceeds outward on the spiral and a New Product Development Project commences.
- • The new product may evolve to represent Product Enhancement Project.

Features of Spiral Model:

- • Realistic approach to the development of large scale-systems and software.
- • Because the software evolves as the process progresses, the developer and customer better understand and react to risks at each evolutionary level.
- • Enables the developer to apply the prototyping approach at any stage in the evolution of the product.
- • It maintains the systematic stepwise approach of classic life cycle but in incorporates it into an iterative framework that realistically reflects the real world.
- • It demands direct consideration of technical risks at all stages of the project and if applied properly, reduce risks before they become problematic.

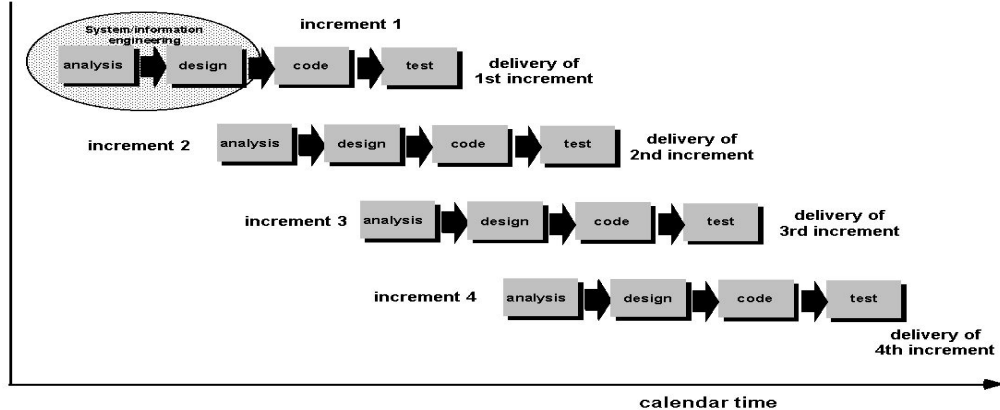
Incremental models

The incremental model applies linear sequences in a staggered fashion as calendar time progresses. Each linear sequence produces deliverable —incrementsl of the software in a manner that is similar to the increments produced by an evolutionary process flow.

- • When an incremental model is used, the first increment is often a core product.
- • The core product is used by the customer (or undergoes detailed evaluation). As a result of use and/or evaluation, a plan is developed for the next increment. The plan addresses the

modification of the core product to better meet the needs of the customer and the delivery of additional features and functionality.

- This process is repeated following the delivery of each increment, until the complete product is produced.



Advantages of Incremental model:

- Generates working software quickly and early during the software life cycle.
- This model is more flexible – less costly to change scope and requirements.
- It is easier to test and debug during a smaller iteration.
- In this model customer can respond to each built.
- Lowers initial delivery cost.
- Easier to manage risk because risky pieces are identified and handled during it's iteration.

Disadvantages of Incremental model:

- Needs good planning and design.
- Needs a clear and complete definition of the whole system before it can be broken down and built incrementally.
- Total cost is higher than waterfall.

21 Define Agility. List any five principles of agility (Apr 19, 21)

Developing software is hard, which is why only an estimated 39% of software projects succeeded in 2012, in the sense that they were delivered on time, within budget and with the required functionality, according to a study of 50,000 IT projects by The Standish Group. Software development is difficult for a lot of reasons; complex technologies, demanding clients, and tight schedules can all combine to make it a challenge. One particularly big reason they fail, though, is because project managers fail to consider and deal with uncertainty.

Uncertainties in software development can come from any number of sources, like the technology itself (Is it new or new to your team?), the client (Will the requirements change on the fly?) and the team itself (Have they worked together before?). Dealing with the inevitable uncertainties properly can be critical to the success of a project. As Jeff Sutherland, the creator of the Scrum methodology has written, for example, allocating time to deal with unplanned-for interruptions is critical to success by Agile teams. But how should software project managers best deal these unknowns?

Identify the sources of uncertainty in the project.

Uncertainty can come from any number of sources. The authors argue that it's important to understand where they're likely to come from in the project at hand. Knowing where uncertainties originate can help PMs more quickly recognize when unexpected things occur during the project and address them promptly. They identify four general sources of

uncertainty: technological (the biggest source, e.g., mixing old and new technologies), the market (client needs, suppliers, partners, etc.), the environment (governments, team size, resources, etc.) and socio-human (e.g., How do your team members learn and use information?).

Use the right project management method for the project.

The authors argue that, when it comes to project management methods, one size does not fit all. Depending on the level of uncertainty in the project goals and solutions, one project management methodology may be preferred over another. For example, if the goals and solutions are both well defined, traditional project management (waterfall) and Agile work well. If the goals are clear but the solution isn't, they recommend Agile. However, if both the goals and the solutions are very uncertain (e.g., an R&D project), Extreme project management is a good choice.

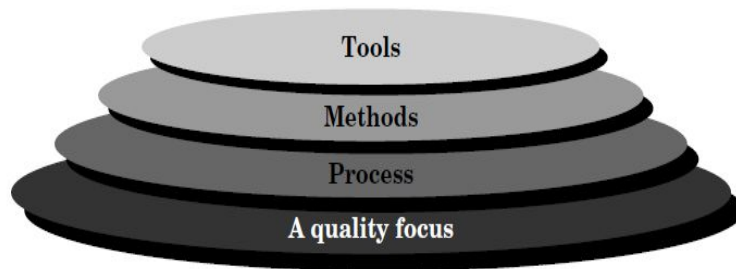
Address unexpected events as soon as possible.

A critical method for dealing with uncertainties, in the authors view, at least, is being vigilant about identifying and dealing with unexpected things as early as possible. Everyone on the team, not just the PM, should constantly be on the lookout for the unexpected. This echoes another recommendation by Sutherland that Scrum teams should address bugs as quickly as possible and not let them linger.

Apply best practices to reduce uncertainty.

Uncertainty can't be eliminated, but it can be reduced. The authors present a number of best practices gleaned from their review of the literature to reduce uncertainty. These include things like managing client expectations, having good communications (both with the client and with the project team), building trust and using flexible contracts, which helps to mitigate the resistance to inevitable changes in the project. They also stressed the importance of being flexible, creative and open to experimentation and improvisation. As they wrote:



22 i) Draw the layered architecture of software engineering. (Nov 18)



ii) What are the merits and demerits of using formal methods for developing a software

Advantages	Dis Advantages
1. Discovers ambiguity, incompleteness, and inconsistency in the software. 2. Offers defect-free software. 3. Incrementally grows in effective solution after each iteration. 4. Incrementally grows in effective solution after each iteration. 5. This model does not involve high complexity rate. Formal specification language semantics	1. Time consuming and expensive. 2. Difficult to use this model as a communication mechanism for non technical personnel. 3. Extensive training is required since only few developers have the essential knowledge to implement this model.

	verify self-consistency.	
23	<p>Assume that you are the technical manager of a software development organization. A client approached you for a software solution. The problems stated by the client have uncertainties which lead to loss if it is not planned and solved. What software development model you will suggest for this project? Justify. Explain that model with a neat sketch along with its pros and cons. (Nov 18)</p> <p>Agile methods of software development are iterative approaches where the software is developed and delivered to customers in increments. Unlike plan-driven approaches, the functionality of these increments is not planned in advance but is decided during the development. The decision on what to include in an increment depends on progress and on the customer's priorities.</p> <ol style="list-style-type: none">1. Although the idea of customer involvement in the development process is an attractive one, its success depends on having a customer who is willing and able to spend time with the development team and who can represent all system stakeholders. Frequently, the customer representatives are subject to other pressures and cannot take full part in the software development.2. Individual team members may not have suitable personalities for the intense involvement that is typical of agile methods, and therefore not interact well with other team members.3. Prioritizing changes can be extremely difficult, especially in systems for which there are many stakeholders. Typically, each stakeholder gives different priorities to different changes.4. Maintaining simplicity requires extra work. Under pressure from delivery schedules, the team members may not have time to carry out desirable system simplifications.5. Many organizations, especially large companies, have spent years changing their culture so that processes are defined and followed. It is difficult for them to move to a working model in which processes are informal and defined by development teams.	
UNIT II REQUIREMENTS ANALYSIS AND SPECIFICATION		
Software Requirements: Functional and Non-Functional, User requirements, System requirements, Software Requirements Document – Requirement Engineering Process: Feasibility Studies, Requirements elicitation and analysis, requirements validation, requirements management-Classical analysis: Structured system Analysis, Petri Nets- Data Dictionary.		
UNIT-II/ PART-A		
1	What are User Requirements and System Requirements? <ul style="list-style-type: none">• User requirements are statements, in a natural language, of what services the system is expected to provide and the constraints under which it must operate.• System requirements set out the system's functions, services and operational constraints in detail. It should define exactly what is to be implemented. It may be part of the contract between the system buyer and the software developers.	
2	Define functional and non-functional requirements. (Nov 10, 14 , Apr 11, 12, 19) <p>The functional requirements of a system describe what the system should do. These requirements depend on the software being developed, the expected users of the software and the general approach taken by the organization when writing requirements.</p> <p>Non-functional requirements are requirements that are not directly concerned with the specific functions delivered by the system. They relate to system properties such as:</p> <p>1)Reliability 2) Response time 3) Storage capacity 4) Availability 5) capability of I/O device</p>	

	6) Data representation in system interfaces 7) System performance 8) Security
3	<p>List two advantages of using traceability tables in the requirements management phase. (Nov 13)</p> <p>These traceability tables are maintained as part of a requirements database so that they may be quickly searched to understand how a change in one requirement will affect different aspects of the system to be built.</p>
4	<p>Draw the DFD notations for a) External entity b) Data items. (Nov 11)</p> <p>Data items-labeled arrows(→)  External entity-boxes ()</p>
5	<p>Define requirements validation.</p> <p>It examines the specifications to ensure that all system requirements have been stated unambiguously by detecting and correcting the inconsistencies, omissions, and errors. It also ensures whether the work products conform to the standards established for the process, the project, and the product.</p>
6	<p>What are the processes involved in requirement engineering? (May 12)</p> <ul style="list-style-type: none"> • Requirements elicitation. • Requirements analysis & negotiation. • Requirements specification. • System modeling. • Requirements validation. • Requirements management.
7	<p>What are the types of viewpoints?</p> <p>Viewpoints can be used as a way of classifying stakeholders and other sources of requirements. There are three generic types of viewpoint:</p> <ul style="list-style-type: none"> • Interactor viewpoints represent people or other systems that interact directly with the system. • Indirect viewpoints represent stakeholders who do not use the system themselves but who influence the requirements in some way. • Domain viewpoints represent domain characteristics and constraints that influence the system requirements.
8	<p>Give the disadvantages of requirements elicitation.</p> <ul style="list-style-type: none"> • Problems of scope: The requirements may address too little or too much informations. • Problems of understanding: Different stakeholders have different requirements, which they may express in different ways. Requirements engineers have to consider all potential sources of requirements and discover commonalities and conflict. • Problems of volatility: It represents the changing nature of the requirements.
9	<p>Define requirements management.</p> <p>It is a set of activities that help the project team to identify, control, and track requirements and changes to requirements at any times as the project proceeds.</p>
10	<p>Define requirement analysis.</p> <p>It is a software engineering task that bridges the gap between system level requirements engineering and software design.</p>
11	<p>How the software requirements analysis can be divided?</p> <p>The software requirements analysis can be divided into the following:</p> <ul style="list-style-type: none"> • Problem recognition. • Evaluation and synthesis. • Modeling. • Specification and Review.
12	Define Traceability

	Traceability is the overall property of requirements specification which reflects the ease of finding related requirements. Three types of traceability information to be maintained are: <ul style="list-style-type: none"> • Source traceability information • Requirement traceability information • Design traceability information
13	Mention some of the Notations for requirements specification. <ul style="list-style-type: none"> • Structured natural language: Use standard form or Templates. • Design description language: Programming language is used. • Graphical notation: Text annotation is used. • Mathematical Specifications: Based on finite state machines or sets.
14	Name three generic classes of methods and tools used in prototyping. Fourth generation techniques, Reusable software components, Formal specification and prototyping environments.
15	Define Requirements engineering. Requirement Engineering provides the appropriate mechanism for understanding what the customer wants, analyzing need, assessing feasibility, negotiating a reasonable solution, specifying the solution unambiguously, validating the specification and managing the requirements as they are transformed into an operational system.
16	What are the different types of checks carried out during Requirement Validation? (or) How are the requirements validated?(Apr 15) <ul style="list-style-type: none"> • Validity checks • Consistency checks • Completeness checks • Realism checks and Verifiability.
17	Name the three objectives used in analysis model. To described what the customer requires, To establish a basis for the creation of a software design, To define a set of requirement that can be validated once the software is built.
18	What is the need for feasibility analysis?(Apr 15) The aim of a feasibility study is to find out whether the system is worth implementing within the given budget and schedule. The purpose of feasibility study is not to solve the problem, but to determine whether the problem is worth solving. This helps to decide whether to proceed with the project or not.
19	Give the two uses of data flow diagram (DFD). Or How does the data flow diagram help in design of software system (Apr 19) <ul style="list-style-type: none"> • To provide an indication of how data are transformed as they move through the system. • To depict the functions that transforms the data flow.
20	Define data flow diagram. It is a graphical representation that depicts information flow and the transforms that are applied as data move from input to output. It may be used to represent a system or software at any level of abstraction.
21	What is data dictionary? What are the informations contained in data dictionary? (Nov 14, 16 , Apr 17) <ul style="list-style-type: none"> • Provides definitions for all elements in the system which include: <ol style="list-style-type: none"> a. Meaning of data flows and stores in DFDs b. Composition of the data flows e.g. customer address breaks down to street number, street name, city and postcode

	<p>c. Composition of the data in stores e.g. in Customer store include name, date of birth, address, credit rating etc.</p> <p>d. Details of the relationships between entities</p> <p>Informations in data dictionary: Name, Alias, Where-used/how- used, Content description and Supplementary information.</p>
22	<p>Give some guidelines to derived data flow diagram?</p> <ul style="list-style-type: none"> • The level 0 data flow diagram should depict the software/system as a single bubble. • Primary input and output should be carefully noted. • Refinement should begin by isolating candidate processes, data objects, and stores to be represented at the next level. • All arrows and bubbles should be labeled with meaningful names. • Information flow continuity must be maintained from level to level. • One bubble at a time should be refined.
23	<p>Give the set of guidelines principles for requirement engineering.</p> <ul style="list-style-type: none"> • Understand the problem before beginning the analysis model. • Develop prototypes that enable a user to understand how human/machine interaction will occur. • Record the origin of and the reason for each and every requirements. • Use multiple views of requirements. • Rank the requirements and eliminate the ambiguity.
24	<p>What are Petri Nets?(Apr 17, Nov 18)</p> <p>Petri net is a directed bipartite graph, in which the nodes represent transitions (i.e. events that may occur, signified by bars) and places (i.e. conditions, signified by circles). The directed arcs describe which places are pre- and/or post conditions for the transitions (signified by arrows).</p>
25	<p>Define throwaway prototyping.</p> <p>A prototype serves solely as a rough demonstration of requirements.</p> <p>It is a close-ended approach.</p>
26	<p>What are the linkages between data flow and ER Diagram? (Apr 16)</p> <p>DFD and ERD are different data models that are mainly used for organizing business data for proper communication between members of a group.</p> <p>DFD shows how data enter a system, are transformed in that system, and how it is stored in it. Meanwhile, ERD represents the entity model and will show what a system or a database will look like but not explain how to implement it.</p>
27	<p>List the characteristics of a good SRS. (Apr 16, 21)</p> <ul style="list-style-type: none"> • Correct • Unambiguous • Complete • Consistent • Ranked for importance and/or stability • Verifiable • Modifiable • Traceable
28	<p>Classify the following as functional/non-functional requirements for a banking system. (Nov 16)</p> <p>a) Verifying bank balance.</p>

	b) Withdrawing money from bank.-FR c) Completion of transactions in less than one second. d) Extending the system by providing more tellers for customers.									
	Functional Requirements	Non-functional Requirements								
	Withdrawing money from bank	Verifying bank balance								
	Completion of transactions in less than one second	Extending the system by providing more tellers for customers								
29	What are the various types of traceability in software engineering? (Apr 18) <ul style="list-style-type: none">• Source traceability: These are basically the links from requirement to stake holders.• Requirement traceability: These are links between dependant requirements• Design traceability: These are links from requirements to design.									
30	Compare prototyping approaches in a software process. (Apr 18) <ul style="list-style-type: none">• Evolutionary Prototyping- the initial prototype is prepared and it is then refined through number of stages to final stage.• Throw-away prototyping-a rough practical implementation of the system is produced. The requirement problems can be identified from this implementation.									
31	Differentiate between normal and exciting requirements.(Apr 17) <table><tr><td>Normal Requirements</td><td>Exciting Requirements</td></tr><tr><td>The objectives and goals that are stated for a product or system during meetings with the customer.</td><td>These features go beyond the customer's expectations.</td></tr><tr><td>If these requirements are present, the customer is satisfied.</td><td>Prove to be very satisfying when presence of the requirement.</td></tr><tr><td>Examples- Requested types of graphical displays, Specific System functions, defined performance.</td><td>Examples-multi touch screen in smart phone, visual voice mail, provides unexpected performance and storage</td></tr></table>		Normal Requirements	Exciting Requirements	The objectives and goals that are stated for a product or system during meetings with the customer.	These features go beyond the customer's expectations.	If these requirements are present, the customer is satisfied.	Prove to be very satisfying when presence of the requirement.	Examples- Requested types of graphical displays, Specific System functions, defined performance.	Examples-multi touch screen in smart phone, visual voice mail, provides unexpected performance and storage
Normal Requirements	Exciting Requirements									
The objectives and goals that are stated for a product or system during meetings with the customer.	These features go beyond the customer's expectations.									
If these requirements are present, the customer is satisfied.	Prove to be very satisfying when presence of the requirement.									
Examples- Requested types of graphical displays, Specific System functions, defined performance.	Examples-multi touch screen in smart phone, visual voice mail, provides unexpected performance and storage									
32	Define quality function development(QFD)(Nov 17) <p>Quality function deployment (QFD) is a quality management technique that translates the needs of the customer into technical requirements for software. QFD "concentrates on maximizing customer satisfaction from the software engineering process".</p>									
33	Draw a usecase diagram for an online shopping which should provide provisions for registering,authenticating the customers an also for online payment through and payment gateway like paypal.(Nov 17 , 18) <div><pre>graph LR subgraph "«Subsystem» Online Shopping" ViewItems((View Items)) MakePurchase((Make Purchase)) Checkout((Checkout)) ClientRegister((Client Register)) ViewItems -- "include" --> MakePurchase MakePurchase -- "include" --> Checkout end WebCustomer[Web Customer] --- ClientRegister NewCustomer[New Customer] --- ClientRegister RegisteredCustomer[Registered Customer] --- ViewItems RegisteredCustomer --- MakePurchase RegisteredCustomer --- Checkout Auth[«Service» Authentication] --- ViewItems Auth --- MakePurchase IdentityProvider[Identity Provider] --- Checkout CreditPayment[Credit Payment Service] --- Checkout PayPal[PayPal] --- Checkout</pre></div>									

34	<p>State two advantages of using Petri Nets (Apr 19)</p> <ul style="list-style-type: none"> Petrinets can be used for design It possesses the expressive power necessary for specifying synchronization aspects of real-time systems.
UNIT-II/ PART-B	
1.	<p>(i) What is the purpose of feasibility study? (Apr 17 ,Nov 17)</p> <p>(ii) State the 'inputs and results of the feasibility study.</p> <p>(iii) List any four issues addressed by a feasibility study.</p> <p>(iv) Elaborate the phases involved when carrying out a feasibility study. (Nov 13)</p> <p>(i) The aims of a feasibility study are to find out whether the system is worth implementing and if it can be implemented, given the existing budget and schedule.</p> <p>The purpose of feasibility study is not to solve the problem, but to determine whether the problem is worth solving. This helps to decide whether to proceed with the project or not.</p> <p>(ii) The input to the feasibility study is a set of preliminary business requirements, an outline description of the system and how the system is intended to support business processes. The results of the feasibility study should be a report that recommends whether or not it is worth carrying on with the requirements engineering and system development process.</p> <p>(iii)</p> <ul style="list-style-type: none"> Gives focus to the project and outline alternatives. Narrows business alternatives Identifies new opportunities through the investigative process. Identifies reasons not to proceed. Enhances the probability of success by addressing and mitigating factors early on that could affect the project. Provides quality information for decision making. Provides documentation that the business venture was thoroughly investigated. Helps in securing funding from lending institutions and other monetary sources. Helps to attract equity investment. <p>The feasibility study is a critical step in the business assessment process. If properly conducted, it may be the best investment you ever made</p> <p>(iv) Carrying out a feasibility study involves information assessment, information collection and report writing.</p>
2.	<p>Explain functional and non-functional requirements in detail.(Nov 14, 17 , Apr 21)</p> <p>Functional requirements</p> <ul style="list-style-type: none"> Statements of services the system should provide, how the system should react to particular inputs and how the system should behave in particular situations. Describe functionality or system services Depend on the type of software, expected users and the type of system where the software is used Functional user requirements may be high-level statements of what the system should do but functional system requirements should describe the system services in detail <p>Examples of functional requirements</p> <ul style="list-style-type: none"> The user shall be able to search either all of the initial set of databases or select a subset from it. The system shall provide appropriate viewers for the user to read documents in the document store.

Requirements Imprecision:

Problems arise when requirements are not precisely stated

Ambiguous requirements may be interpreted in different ways by developers and users

Consider the term 'appropriate viewers'

- User intention - special purpose viewer for each different document type
- Developer interpretation - Provide a text viewer that shows the contents of the document

Requirements completeness and consistency:

In principle requirements should be both complete and consistent

Complete

- They should include descriptions of all facilities required

Consistent

- There should be no conflicts or contradictions in the descriptions of the system facilities

In practice, it is impossible to produce a complete and consistent requirements document

Non-functional requirements

- constraints on the services or functions offered by the system such as timing constraints, constraints on the development process, standards, etc.
- Define system properties and constraints e.g. reliability, response time and storage requirements. Constraints are I/O device capability, system representations, etc.
- Process requirements may also be specified mandating a particular CASE system, programming language or development method
- Non-functional requirements may be more critical than functional requirements. If these are not met, the system is useless

Non-functional Classification**Product requirements**

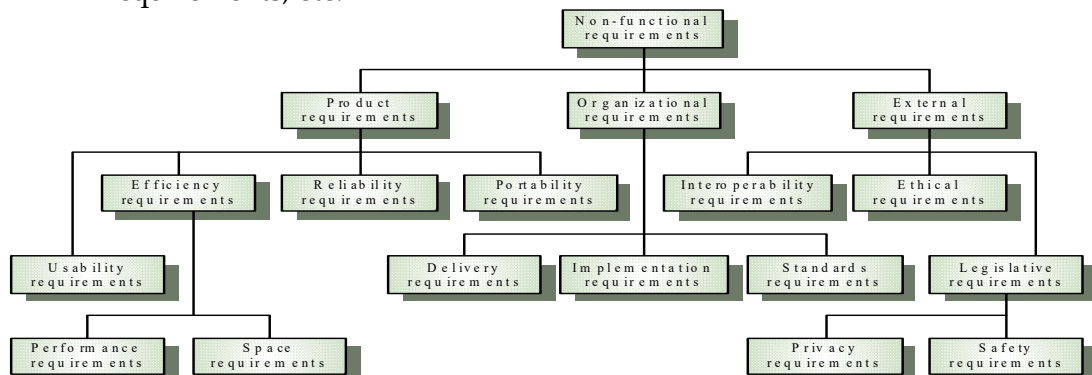
- Requirements which specify that the delivered product must behave in a particular way e.g. execution speed, reliability, etc.

Organisational requirements

- Requirements which are a consequence of organisational policies and procedures e.g. process standards used, implementation requirements, etc.

External requirements

- Requirements which arise from factors which are external to the system and its development process e.g. interoperability requirements, legislative requirements, etc.

**Examples of non-functional requirements:**

Product requirement

- It shall be possible for all necessary communication between the APSE and the user to be expressed in the standard Ada character set

Organisational requirement

- The system development process and deliverable documents shall conform to the process and deliverables defined in XYZCo-SP-STAN-95

External requirement

- The system shall not disclose any personal information about customers apart from their name and reference number to the operators of the system

Goals and requirements:

Non-functional requirements may be very difficult to state precisely and imprecise requirements may be difficult to verify.

Goal

- A general intention of the user such as ease of use

Verifiable non-functional requirement

- A statement using some measure that can be objectively tested

Goals are helpful to developers as they convey the intentions of the system users

Requirement measures:

Property	Measure
Speed	Processed trasactions/second user/Event response time Screen refresh time
Size	K Bytes Number of RAM chips
Ease of Use	Training time Number of help frames
Reliability	Mean time to failure Probability of unavailability Rate of failure occurrence Availability
Robustness	Time to restart after failure Percentage of events causing failure Probability of data corruption on failure
Probability	Percentage of target dependent statement Number of target systems

Requirements Interaction:

Conflicts between different non-functional requirements are common in complex systems

Spacecraft system

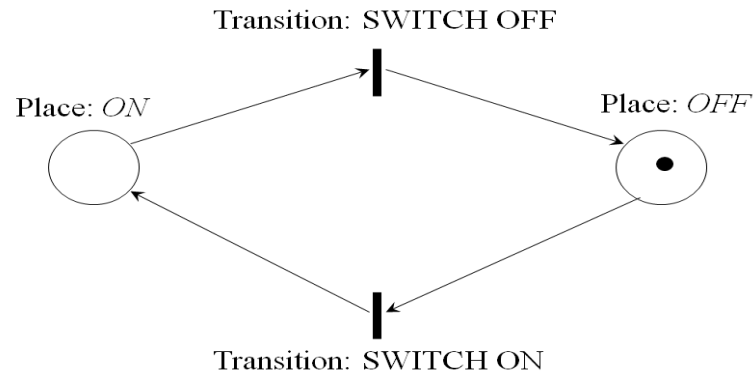
- To minimise weight, the number of separate chips in the system should be minimised
- To minimise power consumption, lower power chips should be used

However, using low power chips may mean that more chips have to be used.

3. Explain the execution of seven distinct functions accomplished in requirement engineering process. (*Nov 10, 12, 14, Apr 11, 13, 15, 17*)
 - **Seven distinct tasks**
 - ✓ **Inception**
During inception, the requirements engineer asks a set of questions to establish...
 - A basic understanding of the problem
 - The people who want a solution

	<ul style="list-style-type: none"> ➤ The nature of the solution that is desired ➤ The effectiveness of preliminary communication and collaboration between the customer and the developer ✓ Elicitation Elicitation may be accomplished through two activities ➤ Collaborative requirements gathering ➤ Quality function deployment ✓ Elaboration During elaboration, the software engineer takes the information obtained during inception and elicitation and begins to expand and refine it Elaboration focuses on developing a refined technical model of software functions, features, and constraints ✓ Negotiation During negotiation, the software engineer reconciles the conflicts between what the customer wants and what can be achieved given limited business resources Requirements are ranked (i.e., prioritized) by the customers, users, and other stakeholders Risks associated with each requirement are identified and analyzed ✓ Specification A specification is the final work product produced by the requirements engineer It is normally in the form of a software requirements specification It serves as the foundation for subsequent software engineering activities It describes the function and performance of a computer-based system and the constraints that will govern its development ✓ Validation <ul style="list-style-type: none"> ➤ During validation, the work products produced as a result of requirements engineering are assessed for quality ➤ The specification is examined to ensure that <ul style="list-style-type: none"> – all software requirements have been stated unambiguously – inconsistencies, omissions, and errors have been detected and corrected – the work products conform to the standards established for the process, the project, and the product ➤ The formal technical review serves as the primary requirements validation mechanism <ul style="list-style-type: none"> – Members include software engineers, customers, users, and other stakeholders ✓ Requirements Management <ul style="list-style-type: none"> ➤ During requirements management, the project team performs a set of activities to identify, control, and track requirements and changes to the requirements at any time as the project proceeds ➤ Each requirement is assigned a unique identifier ➤ The requirements are then placed into one or more traceability tables • Some of these tasks may occur in parallel and all are adapted to the needs of the project • All strive to define what the customer wants • All serve to establish a solid foundation for the design and construction of the software.
4.	<p>Explain Petri Nets in detail.</p> <ul style="list-style-type: none"> ➤ A Petri Nets (PN) comprises places, transitions, and arcs <ul style="list-style-type: none"> – Places are system states – Transitions describe events that may modify the system state

- Arcs specify the relationship between places
- Tokens reside in places, and are used to specify the state of a PN

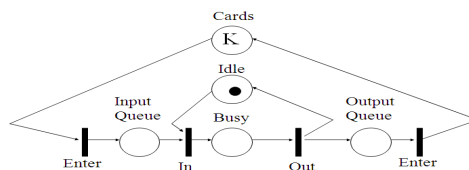


- Two places: Off and On
- Two transitions: Switch Off and Switch On
- Four arcs
- The off condition is true
- A transition can fire if an input token exists
 - One token is moved from the input place to the output place.

PN properties

- 8-tuple mathematical model
 - $M = \{P, T, I, O, H, PAR, PRED, MP\}$
 - P - the set of places
 - T - the set of transitions
 - I, O, H - Input, output, inhibition function
 - PAR - the set of parameters
 - PRED - Predicates restricting parameter range
 - PM - Parameter value
- From this linear algebra can be used to analyze a network

Manufacturing Example



- Very rich modeling
- Easily capable of modeling software project, requirements, architectures, and processes
- Drawbacks
 - Complex rules
 - Analysis quite complex

5. What is data dictionary? Explain with an example. (Apr 11)

- Provides definitions for all elements in the system which include:
 - Meaning of data flows and stores in DFDs
 - Composition of the data flows e.g. customer address breaks down to street number, street name, city and postcode

	<p>g. Composition of the data in stores e.g. in Customer store include name, date of birth, address, credit rating etc.</p> <p>h. Details of the relationships between entities</p> <p>➤ Data dictionary Notation</p> <p>= is composed of</p> <p>+ and</p> <p>() optional (may be present or absent)</p> <p>{ } iteration</p> <p>[] select one of several alternatives</p> <p>** comment</p> <p>@ identifier (key field) for store</p> <p> separates alternative choices in the [] construct</p> <p>➤ Data dictionary Examples</p> <p>name = courtesy-title + first-name + (middle-name) + last-name</p> <p>courtesy-title = [Mr. Miss Mrs. Ms. Dr. Professor]</p> <p>first-name = {legal-character}</p> <p>middle-name = {legal-character}</p> <p>last-name = {legal-character}</p> <p>legal-character = [A-Z a-z 0-9 ' -]</p> <p>Current-height = ** *units: metres; range: 1.00-2.50*</p> <p>sex = ***values: [M F]*</p> <p>As both are elementary data, no composition need be shown, though an explanation of the relevant units/symbols is needed order = customer-name + shipping-address + 1{item}10 means that an order always has a customer name and a shipping address and has between 1 and 10 items.</p>
6.	<p>How does the analysis modeling help to capture unambiguous and consistent requirements? Discuss several methods for requirements validation. (Nov 11)</p> <p>Goals of Analysis Modeling</p> <ul style="list-style-type: none"> ➤ Provides the first technical representation of a system ➤ Is easy to understand and maintain ➤ Deals with the problem of size by partitioning the system ➤ Uses graphics whenever possible ➤ Differentiates between essential information versus implementation information ➤ Helps in the tracking and evaluation of interfaces ➤ Provides tools other than narrative text to describe software logic and policy ➤ Flow-oriented modeling – provides an indication of how data objects are transformed by a set of processing functions ➤ Scenario-based modeling – represents the system from the user's point of view ➤ Class-based modeling – defines objects, attributes, and relationships ➤ Behavioral modeling – depicts the states of the classes and the impact of events on these states <p>Requirements validation</p> <ul style="list-style-type: none"> ➤ Concerned with demonstrating that the requirements define the system that the customer really wants. ➤ Requirements validation covers a part of analysis in that it is concerned with finding problems with requirements. ➤ Requirements error costs are high so validation is very important <ul style="list-style-type: none"> ○ Fixing a requirements error after delivery may cost up to 100 times the cost of fixing an implementation error. ○ In fact, a change to the requirements usually means that the system design

and the implementation must also be changed and the testing has to be performed again.

Checks required during the requirements validation process

- **Validity checks.** Does the system provide the functions which best support the customer's needs? (Other functions maybe identified by a further analysis)
- **Consistency checks.** Are there any requirements conflicts?
- **Completeness checks.** Are all the requirements needed to define all functions required by the customer sufficiently specified?
- **Realism checks.** Can the requirements be implemented given available budget, technology and schedule?
- **Verifiability.** Can the requirements be checked?

Requirements validation techniques

The following techniques can be used individually or in conjunction.

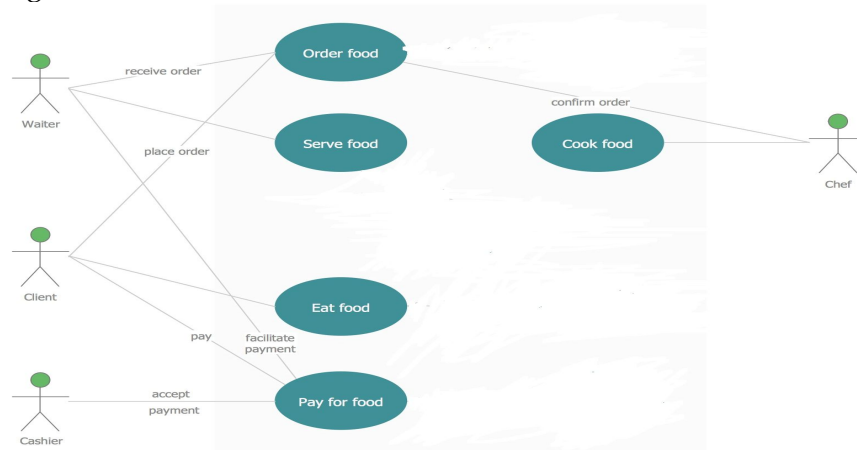
- **Requirements reviews**
 - Systematic manual analysis of the requirements performed by a team of reviewers
- **Prototyping**
 - Using an executable model of the system to check requirements.
- **Test-case generation**
 - Developing tests for requirements to check testability.

If the test is difficult to design, usually the related requirements are difficult to implement.

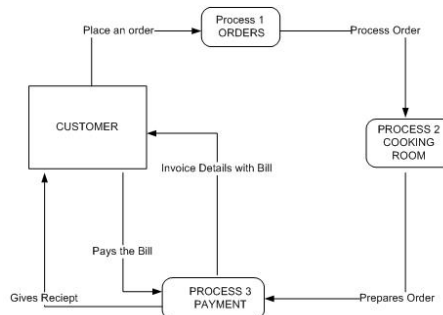
7. Draw Use Case and Data Flow diagrams for a "Restaurant System". The activities of the Restaurant system are listed below.

Receive the Customer food Orders, Produce the customer ordered foods, Serve the customer with their ordered foods, Collect Payment from customers, Store customer payment details, Order Raw Materials for food products, Pay for Raw Materials for food products, Pay for Raw Materials and Pay for Labor. (Apr 15)

Use Case Diagram

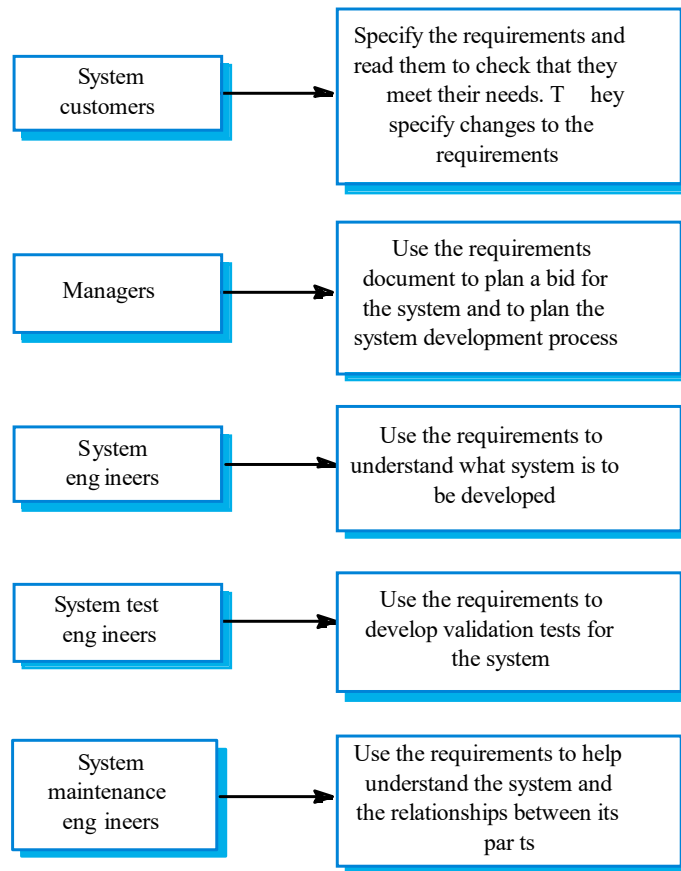


Data Flow Diagram



8. (i) What are the components of the standard structure for the software requirements document? Explain in detail. (Apr 16)
 (ii) Write the software requirement specification for a system of your choice/Train reservation system (Apr 14)(Nov 17)
 (iii) Write the software requirement specification for a system of your choice. (May14) (Apr 21)
 (iv) Write the software requirement specification for a system of your choice. (May14) (Apr 21)

- The requirements document is the official statement of what is required of the system developers.
- Should include both a definition of user requirements and a specification of the system requirements.
- It is NOT a design document. As far as possible, it should set out WHAT the system should do rather than HOW it should do it



- IEEE Standard
- ✓ Defines a generic structure for a requirements document that must be instantiated for each specific system.
 - Introduction.
 - General description.
 - Specific requirements.
 - Appendices.
 - Index.
- ✓ Requirement Document Structure
 - Preface
 - Introduction
 - Glossary
 - User requirements definition
 - System architecture

	<ul style="list-style-type: none"> • System requirements specification • System models • System evolution • Appendices • Index 		
9.	<p>Give an account about data models and explain with appropriate diagrams.</p> <p>Data modeling examines data objects independently of processing, focuses attention on the data domain, creates a model at the customer's level of abstraction and indicates how data objects relate to one another.</p> <p>Data objects</p> <p>Something that is described by a set of attributes and that will be manipulated within the software(system)</p> <ul style="list-style-type: none"> • Each instance of an object can be identified uniquely. • Each plays a necessary role in the system (i.e) the system could not function without access to instance of the object. • Each is described by attribute that are themselves data items <p>Typical objects</p> <ul style="list-style-type: none"> • External entities • Things • Occurrence or events • Roles • Organizational units • Places • Structures <p>Data objects and Attributes</p> <ul style="list-style-type: none"> • A data objects contains set of attributes that act as an aspect, quality, characteristics or descriptor of the object <table border="1" data-bbox="393 1159 721 1423"> <tr> <td>Object : Automobiles</td> </tr> <tr> <td>Attributes: Make Model Body type Price Option code</td> </tr> </table> <p>Relationship</p> <p>Relationship indicates "connectedness"; a fact that must be remembered by the system and cannot or is not computed or derived mechanically</p> <ul style="list-style-type: none"> • Several instance of a relationship can exist • Objects can be related in many different ways <div data-bbox="393 1635 937 1707"> <pre> graph LR Person[Person] --- Car[Car] </pre> </div> <p>(a)A basic connection between data objects</p> <div data-bbox="393 1799 959 1942"> <pre> graph LR subgraph Owns Person1[Person] <--> Car1[Car] end subgraph Insured_to_drive Person2[Person] --> Car2[Car] end </pre> </div>	Object : Automobiles	Attributes: Make Model Body type Price Option code
Object : Automobiles			
Attributes: Make Model Body type Price Option code			

	<p>(b) Relationships between data objects</p> <p>Cardinality It specifies how the number of occurrence of one object are related to the number of occurrence of another object (1:1, 1:N, N:M)</p> <p>Modality Zero (0) – for optional object relationship One (1) – for mandatory relationship</p>
10.	<p>Write the software requirement specification for a system of your choice. (MAY/JUNE 2014)</p> <p>ONLINE TICKET RESERVATION SYSTEM</p> <p>PROBLEM STATEMENT:</p> <p>This project is about online ticket reservation and consists of two modules. The reservation and the cancellation module. The reservation module allows the user to reserve tickets for a particular train on a particular date. If there is a ticket available, the users can know the vacancy details through the enquiry module. The cancellation module allows user to cancel the tickets for a particular date through reservation officer (system). This module performs status reveal before tickets are being reserved and after they get booked. All these modules together prove to be a flexible online reservation system and it provides complete flexibility to end users and it assumes the desired performance.</p> <p>OVERALL DESCRIPTION:</p> <p>MODULES:</p> <ul style="list-style-type: none"> • Login • Display train list • Search for train • Reservation • Cancellation • Train Status <p>MODULE DELIVERABLES:</p> <p>1. LOGIN</p> <p>Basic Flow This use case starts when the passenger wishes to Login to the Online Ticket Reservation system</p> <ul style="list-style-type: none"> • The System requests that the passenger enter his/her name and password • The passenger enters his/her name and password • The System validates the entered name and password and logs the passenger into the System <p>Alternative Flows: Invalid Name/Password If, in the Basic flow, the passenger enters an invalid name and/or password, the system displays an error message. The passenger chooses to either return to the beginning of the Basic flow or cancel the login, at which point the use case ends.</p> <p>Pre-Conditions: None Post-Conditions: If the use case was successful, the passenger is now logged into the system. If not, the system State is unchanged.</p> <p>2. Display Train List</p> <p>Basic Flow: This use case gives passenger information about each train namely train no, train name, Stations passes, Arrival Time, Departure Time etc</p> <p>Alternative Flows: None Pre-Conditions: None</p>

Post-Conditions: If the use case was successful, the passenger information about each train namely train no, train name, Stations passes, Arrival Time, Departure Time etc

3. Search for Train

Basic flow

The passenger can obtain train information either by entering train no or Source and Destination Station

1. If the passenger train no gives the information about train
2. If the passenger enter Source and Destination Station from list gives information about list of trains passing through station. From the list link will be provided to each train, which contains the information

Alternate flow: If the passenger enters an invalid train no then it gives error message invalid train no and asks the passenger to enter a valid train no.

Pre-Conditions: None

Post-Conditions: If the use case was successful, the passenger can able to view the list of trains.

4.Reservation

Basic flow

1. The user reserves the ticket by giving following
 - a) Passenger name, Sex, Age, Address
 - b) Credit Card No, Bank Name
 - c) Class through passenger is going to travel i.e First class or Second class or AC
 - d) Train no and Train name, Date of Journey and number of tickets to be booked.
2. If the ticket is available in a train then the ticket will be issued with PNR No.else the ticket will be issued with a waiting list number.

Alternative flow: If the passenger gives an invalid credit card no or specified a bank where does have any account. Error message will be displayed.

Pre-Conditions: The passenger has to decide about the train he is going to travel.

Post-Conditions: If the use case was successful, the passenger will get the ticket.

5. Cancellation

Basic flow

This use case used by passenger to cancel the ticket, which he/she booked earlier by Entering PNR No. The cancellation has been done reallocating the tickets allotted to the Passenger.

Alternate flow: If the Passenger had entered invalid PNR No then has been asked to enter valid PNR No.

Pre-Conditions: The Passenger had reserved tickets in a train.

Post-Conditions: If the use case was successful, the passenger can cancel the ticket.

6. Ticket Status

Basic flow

1. The passenger should give PNR No to know the status of ticket, which he/she booked earlier.
2. If the PNR No is valid, the status of the ticket will be displayed.

Alternate flow: If passenger had entered an invalid no or PNR NO, which does not exists then error Message will be displayed.

Pre-Conditions: The Passenger had reserved tickets in a train.

Post-Conditions: If the use case was successful, the passenger can view status of the ticket.

Req #	Description	Priority
REQ-S1	The user will be able to search for trains through a	[Priority = High]

	standardized screen. Advanced options will be available by clicking appropriate links.		
	REQ-S2	Through the standard trains search method the user will be	[Priority = Medium]
	searching round trip trains. The search criteria can be modified by the user by selecting one-way and multi- destination options which would be displayed on a new window.		
	REQ-S3	Through the standard trains search method the user shall be	[Priority = High]
	able to specify the departure and return date of their trains.		
	REQ-S4	Through an advanced train search method the user shall be	[Priority = Medium]
	able to specify the arrival train times.		
	REQ-S5	The standard train search method will enable the user to	[Priority = Medium]
	search both precise dates as well as a range of arrival and departure dates.		
11.	REQ-S6	The standard train search method will allow the user to	[Priority = Low]
	specify a preferred railline. This is optional, i.e. the user may or may not specify the railline of preference.		
	REQ-S7	The user will have the option to express a preference of	[Priority = Low]
	What are the types of behavioral models? Explain with examples. (<i>Apr 13, 14</i>)		
	<ul style="list-style-type: none"> ➤ The behavioral model indicates how software will respond to external events ➤ To create the model, you should perform the following steps: <ul style="list-style-type: none"> • Evaluate all use cases to fully understand the sequence of interaction within the system. • Identify events that drive the interaction sequence and understand how these events relate to specific objects. • Create a sequence for each use case. • Build a state diagram for the system. • Review the behavioral model to verify accuracy and consistency. 		
	Identifying Events with the Use Case		
	A use case is examined for points of information exchange.		
	<ul style="list-style-type: none"> • The homeowner uses the keypad to key in a four-digit password. • The password is compared with the valid password stored in the system. • If the password is incorrect, the once and reset itself for additional input. • If the password is control panel will beep correct, the control panel awaits further 		

	<p>action.</p> <ul style="list-style-type: none"> • Once all events have been identified, they are allocated to the objects involved. • Objects can be responsible for generating events. <p>State Representations</p> <ul style="list-style-type: none"> • The state of each class as the system performs its function and • The state of the system as observed from the outside as the system performs its function. <p>Two different behavioral representations are State diagrams for analysis classes that represent active states for each class and the events (triggers) that cause changes between these active states. The second type of behavioral representation, called a sequence diagram in UML, indicates how events cause transitions from object to object.</p>
12.	<p>(i) Differentiate between user and system requirements.</p> <p>ii) Describe the requirement change management process in detail. (<i>Apr 16</i>)</p> <p>User requirements are statements, in a natural language plus diagrams, of what services the system is expected to provide to system users and the constraints under which it must operate.</p> <p>The user requirements for a system should describe the functional and nonfunctional requirements so that they are understandable by system users who don't have detailed technical knowledge. Ideally, they should specify only the external behavior of the system.</p> <p>2. System requirements are more detailed descriptions of the software system's functions, services, and operational constraints. The system requirements document (sometimes called a functional specification) should define exactly what is to be implemented. It may be part of the contract between the system buyer and the software developers.</p> <p>System requirements are expanded versions of the user requirements that are used by software engineers as the starting point for the system design. They add detail and explain how the user requirements should be provided by the system. They may be used as part of the contract for the implementation of the system and should therefore be a complete and detailed specification of the whole system.</p> <p>ii) Describe the requirement change management process in detail. (<i>MAY/JUNE 2016</i>)</p> <p>Requirements change management (Figure 4.18) should be applied to all proposed changes to a system's requirements after the requirements document has been approved. Change management is essential because you need to decide if the benefits of implementing new requirements are justified by the costs of implementation.</p> <p>1. Problem analysis and change specification The process starts with an identified using a formal process for change management is that all change proposals are treated consistently and changes to the requirements document are made in a controlled way. There are three principal stages to a change management process: requirements problem or, sometimes, with a specific change proposal. During this stage, the problem or the change proposal is analyzed to check that it is valid. This analysis is fed back to the change requestor who may respond with a more specific requirements change proposal, or decide to withdraw the request.</p> <p>2. Change analysis and costing The effect of the proposed change is assessed using traceability information and general knowledge of the system requirements. The cost of making the change is estimated both in terms of modifications to the requirements document and, if appropriate, to the system design and implementation. Once this analysis is completed, a decision is made whether or not to proceed with the requirements change.</p> <p>3. Change implementation The requirements document and, where necessary, the system design and implementation, are modified. You should organize the requirements document so that you can make changes to it without extensive rewriting or reorganization. As with programs, changeability in documents is achieved by minimizing external references and</p>

	<p>making the document sections as modular as possible. Thus, individual sections can be changed and replaced without affecting other parts of the document.</p> <p>If a new requirement has to be urgently implemented, there is always a temptation to change the system and then retrospectively modify the requirements document. You should try to avoid this as it almost inevitably leads to the requirements specification and the system implementation getting out of step. Once system changes have been made, it is easy to forget to include these changes in the requirements document or to add information to the requirements document that is inconsistent with the implementation.</p> <p>Agile development processes, such as extreme programming, have been designed to cope with requirements that change during the development process. In these processes, when a user proposes a requirements change, this change does not go through a formal change management process. Rather, the user has to prioritize that change and, if it is high priority, decide what system features that were planned for the next iteration should be dropped.</p>
13.	<p>Write a note on what are the difficulties in elicitation, requirement elicitation (<i>Apr 17, 21</i>)</p> <p>Thin spread of domain knowledge</p> <ul style="list-style-type: none"> ✓ It is rarely available in an explicit form ✓ distributed across many sources ✓ with conflicts between knowledge from different sources <p>Tacit knowledge (The “say-do” problem)</p> <ul style="list-style-type: none"> ✓ People find it hard to describe knowledge they regularly use <p>Limited Observability</p> <ul style="list-style-type: none"> ✓ The problem owners might be too busy coping with the current system ✓ Presence of an observer may change the problem <p>Bias</p> <ul style="list-style-type: none"> ✓ People may not be free to tell you what you need to know ✓ People may not want to tell you what you need to know <p>Requirements Elicitation is one of the most difficult stages of analysis, with numerous communication barriers existing between the analyst and client that make eliciting requirements difficult. Analysts and clients often speak in different general languages, with analysts often being more technical in nature, while clients will often speak more from a business perspective. This makes common understanding difficult. Several other general challenges in requirements elicitation, including conflicting requirements, unspoken or assumed requirements, difficulty in meeting with relevant stakeholders, stakeholder resistance to change, and not enough time set for meeting with all stakeholders.</p> <p>Problems of requirements elicitation can be grouped into three categories:</p> <ul style="list-style-type: none"> ✓ problems of scope, in which the requirements may address too little or too much information; ✓ problems of understanding, within groups as well as between groups such as users and developers; and

- ✓ problems of volatility, i.e., the changing nature of requirements.

The list of ten elicitation problems could be classified according to this framework as follows:

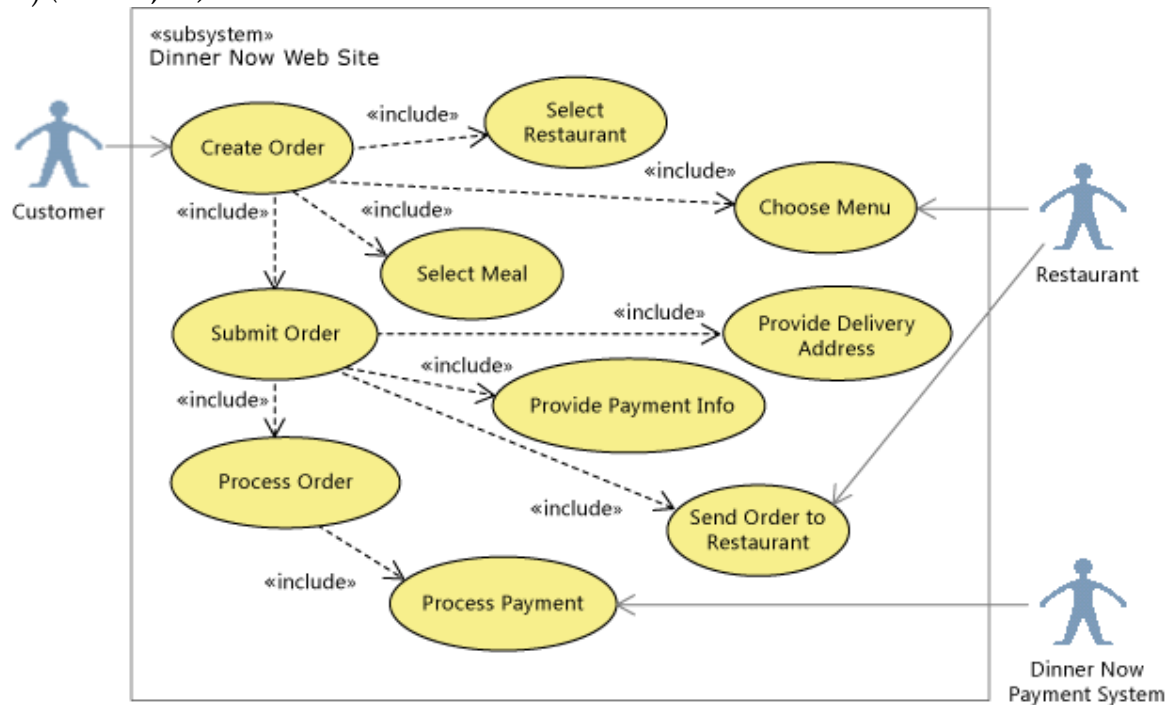
Problems of scope

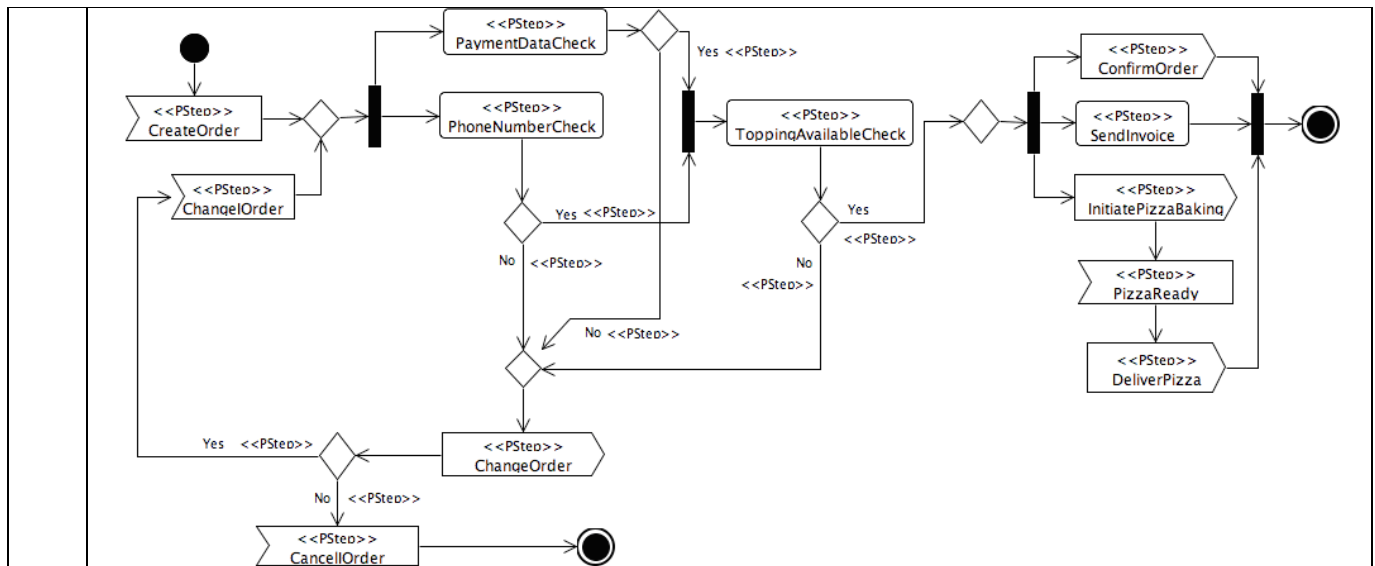
- ✓ the boundary of the system is ill-defined
- ✓ unnecessary design information may be given

Problems of understanding

- ✓ users have incomplete understanding of their needs
- ✓ users have poor understanding of computer capabilities and limitations
- ✓ analysts have poor knowledge of problem domain
- ✓ user and analyst speak different languages
- ✓ ease of omitting "obvious" information
- ✓ conflicting views of different users

14. Consider the process of ordering a pizza over the phone. Draw the use case diagram and also sketch the activity diagram representing each step of the process, from the moment you pick up the phone to the point where you start eating the pizza. Include activities that others need to perform. Add exception handling to the activity diagram you developed. Consider at least two exception (e.g. delivery person wrote down wrong address, deliver person brings wrong pizza) (Nov 17, 18)





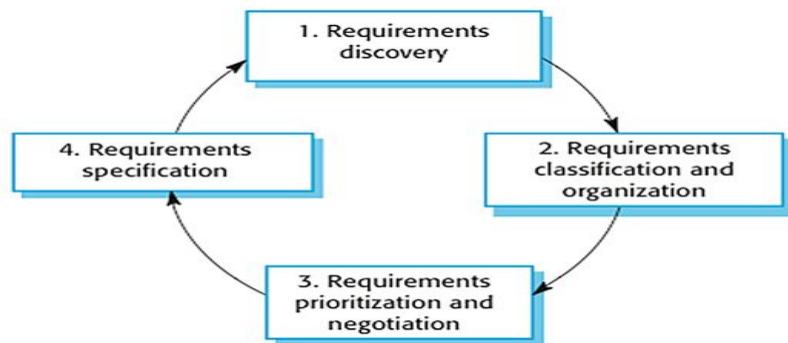
15. What is requirements elicitation? Briefly describe the various activities performed in requirements elicitation phase with an example of a watch system that facilitates set time and alarm. (Apr 18)

Requirements Elicitation is one of the most difficult stages of analysis, with numerous communication barriers existing between the analyst and client that make eliciting requirements difficult. Analysts and clients often speak in different general languages, with analysts often being more technical in nature, while clients will often speak more from a business perspective. This makes common understanding difficult. Several other general challenges in requirements elicitation, including conflicting requirements, unspoken or assumed requirements, difficulty in meeting with relevant stakeholders, stakeholder resistance to change, and not enough time set for meeting with all stakeholders.

The requirements elicitation and analysis has 4 main process

We typically start by gathering the requirements, this could be done through a general discussion or interviews with your stakeholders, also it may involve some graphical notation. Then you organize the related requirements into sub components and prioritize them, and finally, you refine them by removing any ambiguous requirements that may raise from some conflicts.

Here are the 4 main process of requirements elicitation and analysis.



The process of requirements elicitation and analysis

1. Requirements Discovery

It's the process of interacting with, and gathering the requirements from, the stakeholders about the required system and the existing system (if exist).

Interviews

In Interviews, requirements engineering teams put the questions to the stakeholder about the system that's currently used, and the system to be developed, and hence they can gather the requirements from the answers.

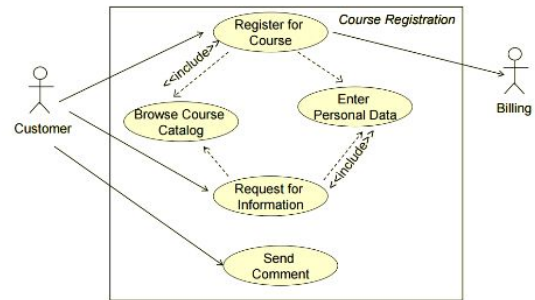
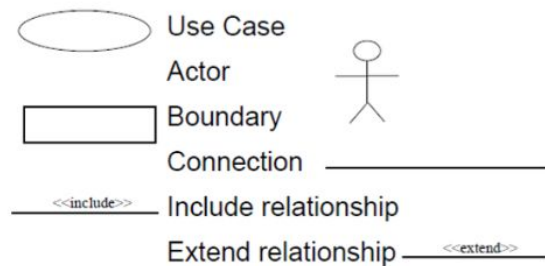
The questions fall under two categories:

1. *Closed-Ended* questions: A pre-defined set of question.
2. *Open-Ended* questions: There is no a pre-defined expected answer; they are more of generic questions. It's used to explore issues that are not clear in a less structured way.

Use Cases & Scenarios

The use cases and scenarios are two different techniques, but, usually they are used together. Use cases identify interactions between the system and it's users or even other external systems (using graphical notations), while a scenario is a textual description of one or more of these interactions.

Use case involves some symbols to describe the system:



Use case diagram symbols and an example

1. **Actors:** Are those who interact with the system; human or other systems
2. **Interaction (Use Case):** It denotes the name of the interaction (verb). It's represented as a named ellipse.
3. **Connection:** Lines that links between the actors and the interactions.
4. **Include Relationship:** It denotes a connection between two interactions when an interaction is invoked by another. As an example, splitting a large interaction into several interactions.
5. **Exclude Relationship:** It denotes a connection between two interactions when you want to extend an interaction by adding an optional behavior, but you can use the main interaction on its own without the extending interaction.

2. Requirements Classification & Organization

It's very important to organize the overall structure of the system. Putting related requirements together, and decomposing the system into sub components of related requirements. Then, we define the relationship between these components.

3. Requirements Prioritization & Negotiation

We previously explained why eliciting and understanding the requirements is not an easy process. One of the reasons is the conflicts that may arise as a result of having different stakeholders involved. *Why?* Because it's hard to satisfy all parties, if it's not impossible. This activity is concerned with prioritizing requirements and finding and resolving requirements conflicts through negotiations until you reach a situation where some of the stakeholders can compromise.

4. Requirements Specification

The requirements are then documented. We'll discuss requirements specification in more detail in "Requirements Engineering – Requirements Specification".

Requirements Elicitation

The process through which the customers, buyers, or users of a software system discover, reveal, articulate, and understand a watch system that facilitates to set time and alarm. For example, gathering requirements based on time zone, framing windows, options to set alarm, etc

Requirements Analysis

The process of reasoning about the requirements that have been elicited; it involves activities such as examining requirements for conflicts or inconsistencies, combining related requirements, and identifying missing requirements. The above said requirements are analyzed properly and details are taken from device calendar database.

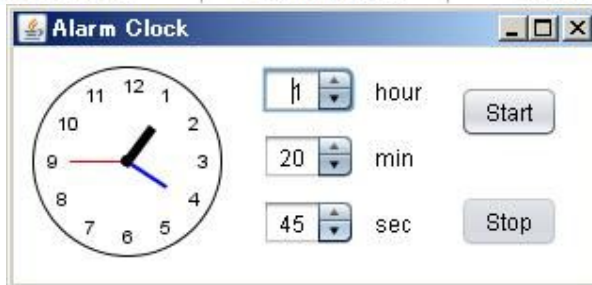
Requirements Specification

The process of recording the requirements in one or more forms, including natural language and formal, symbolic, or graphical representations; also, the product that is the document produced by that process.

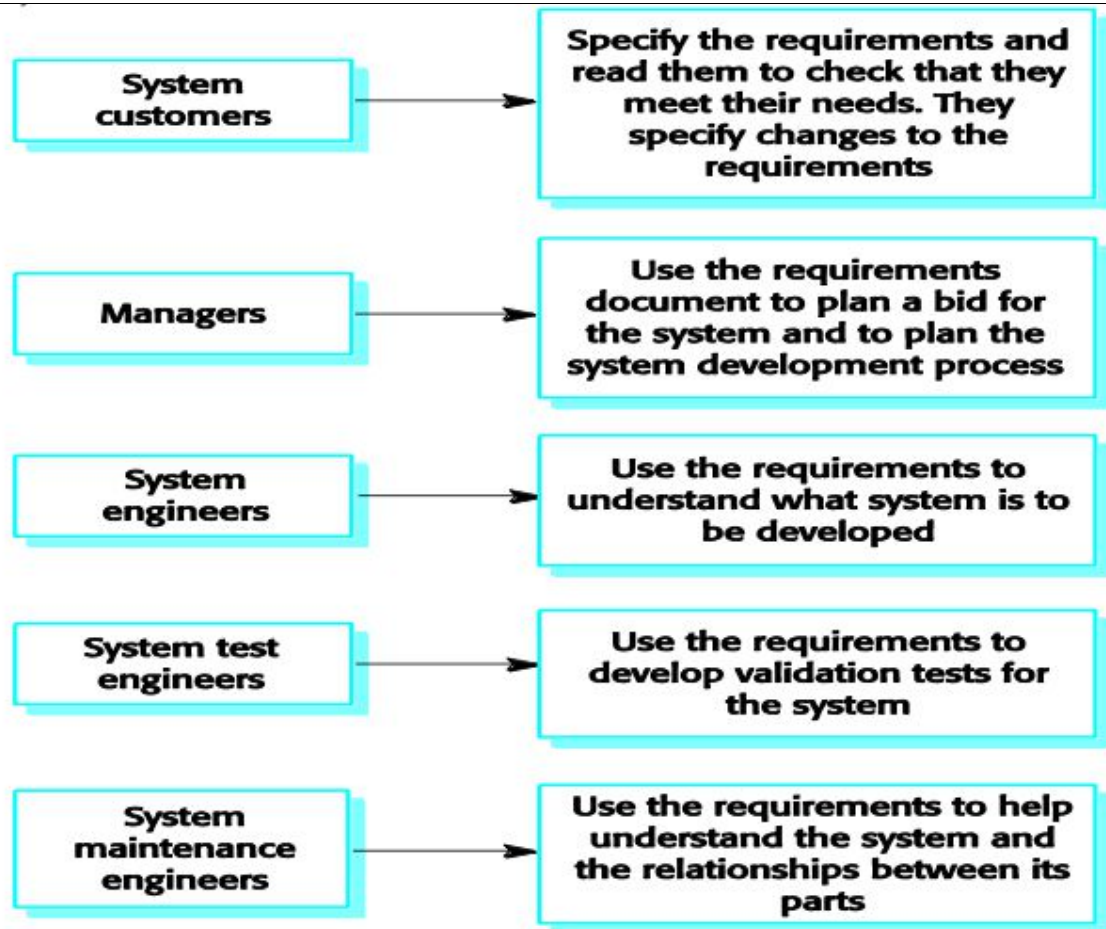
Requirements Validation

The process of confirming with the customer or user of the software that the specified requirements are valid, correct, and complete. The following table represents elicitation and analysis which can be done by design, brainstorming activities, interviews and framework activities.

<i>Technique</i>	<i>Preparatory Step</i>	<i>Implementation Step</i>
Joint Application Design	15 minutes	60 minutes
Brainstorming	15 minutes	60 minutes
Interviewing	35 minutes	40 minutes
PIECES framework	35 minutes	40 minutes



16. What is SRS? Explain in detail the various components of an SRS. (*Apr 18*)
- The requirements document is the official statement of what is required of the system developers.
- Should include both a definition of user requirements and a specification of the system requirements.
 - It is NOT a design document. As far as possible, it should set out WHAT the system should do rather than HOW it should do it.



Defines a generic structure for a requirements document that must be instantiated for each specific system.

- Introduction.
- General description.
- Specific requirements.
- Appendices.
- Index.
- Requirement Document Structure
- Preface
- Introduction
- Glossary
- User requirements definition
- System architecture
- System requirements specification
- System models
- System evolution
- Appendices
- Index

• The information that is included in a requirements document depends on the type of software being developed and the approach to development that is to be used.

REQUIREMENTS SPECIFICATION

- Requirements specification is the process of writing down the user and system requirements in a requirements document. Ideally, the user and system requirements should be clear, unambiguous, easy to understand, complete, and consistent.

	<ul style="list-style-type: none"> • The user requirements for a system should describe the functional and non-functional requirements so that they are understandable by system users who don't have detailed technical knowledge. • System requirements are expanded versions of the user requirements that are used by software engineers as the starting point for the system design. They add detail and explain how the user requirements should be provided by the system. • The system requirements should simply describe the external behavior of the system and its operational constraints. They should not be concerned with how the system should be designed or implemented. • At the level of detail required to completely specify a complex software system, it is practically impossible to exclude all design information. There are several reasons for this: <ul style="list-style-type: none"> ✓ The system requirements are organized according to the different subsystems that make up the system. ✓ Systems must interoperate with existing systems, which constrain the design and impose requirements on the new system. ✓ The use of a specific architecture to satisfy non-functional requirements may be necessary.
17.	<p>i)What is feasibility study? How it helps in requirement engineering process? (Nov 18)</p> <p>ii)Describe the requirement change management process in detail. (Apr 16)</p> <p>iii) List the stakeholders and all types of requirement for online train reservation system.</p> <p>Feasibility study An estimate is made of whether the identified user needs may be satisfied using current software and hardware technologies. The study considers whether the proposed system will be cost-effective from a business point of view and if it can be developed within existing budgetary constraints. A feasibility study should be relatively cheap and quick. The result should inform the decision of whether or not to go ahead with a more detailed analysis.</p> <p>i. How will you classify the requirement types for a project? Give example.</p> <p>User requirements are statements, in a natural language plus diagrams, of what services the system is expected to provide to system users and the constraints under which it must operate.</p> <ol style="list-style-type: none"> 1. The user requirements for a system should describe the functional and nonfunctional requirements so that they are understandable by system users who don't have detailed technical knowledge. Ideally, they should specify only the external behavior of the system. 2. System requirements are more detailed descriptions of the software system's functions, services, and operational constraints. The system requirements document (sometimes called a functional specification) should define exactly what is to be implemented. It may be part of the contract between the system buyer and the software developers. <p>System requirements are expanded versions of the user requirements that are used by software engineers as the starting point for the system design. They add detail and explain how the user requirements should be provided by the system. They may be used as part of the contract for the implementation of the system and should therefore be a complete and detailed specification of the whole system.</p> <p>ii)Describe the requirement change management process in detail. (MAY/JUNE 2016)</p> <p>Requirements change management (Figure 4.18) should be applied to all proposed changes to a system's requirements after the requirements document has been approved. Change management is essential because you need to decide if the benefits of implementing new requirements are justified by the costs of implementation.</p> <ol style="list-style-type: none"> 1. Problem analysis and change specification The process starts with an identified using a formal process for change management is that all change proposals are treated consistently and changes to the requirements document are made in a controlled way. There are three

principal stages to a change management process: requirements problem or, sometimes, with a specific change proposal. During this stage, the problem or the change proposal is analyzed to check that it is valid. This analysis is fed back to the change requestor who may respond with a more specific requirements change proposal, or decide to withdraw the request.

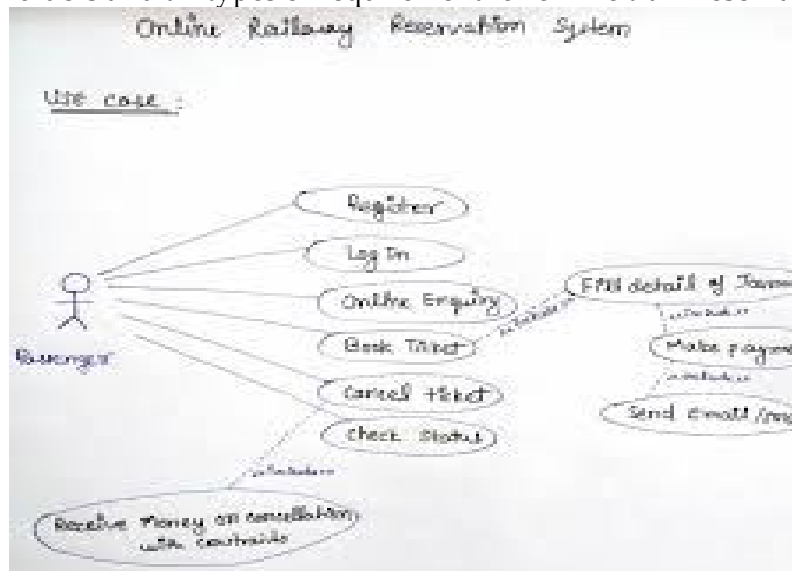
2. Change analysis and costing The effect of the proposed change is assessed using traceability information and general knowledge of the system requirements. The cost of making the change is estimated both in terms of modifications to the requirements document and, if appropriate, to the system design and implementation. Once this analysis is completed, a decision is made whether or not to proceed with the requirements change.

3. Change implementation The requirements document and, where necessary, the system design and implementation, are modified. You should organize the requirements document so that you can make changes to it without extensive rewriting or reorganization. As with programs, changeability in documents is achieved by minimizing external references and making the document sections as modular as possible. Thus, individual sections can be changed and replaced without affecting other parts of the document.

If a new requirement has to be urgently implemented, there is always a temptation to change the system and then retrospectively modify the requirements document. You should try to avoid this as it almost inevitably leads to the requirements specification and the system implementation getting out of step. Once system changes have been made, it is easy to forget to include these changes in the requirements document or to add information to the requirements document that is inconsistent with the implementation.

Agile development processes, such as extreme programming, have been designed to cope with requirements that change during the development process. In these processes, when a user proposes a requirements change, this change does not go through a formal change management process. Rather, the user has to prioritize that change and, if it is high priority, decide what system features that were planned for the next iteration should be dropped.

ii. List the stakeholders and all types of requirement for online train reservation system.



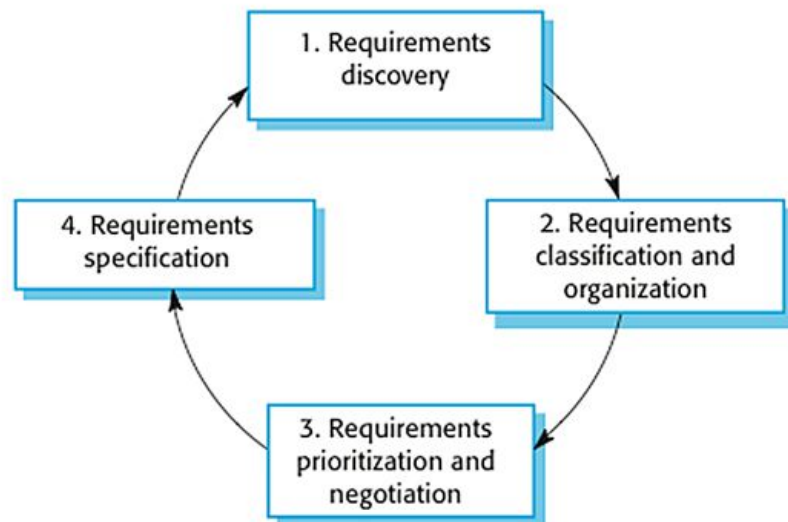
- | | |
|-----|---|
| 18. | Develop the software requirements document for the following requirement. A coffee vending machine serves coffee to customers. A customer can choose a type of coffee among a list often options, supply the amount required and get served. Each coffee is prepared by adding units of hot water, coffee powder, milk and sugar. The receipt for each coffee is stored. (Apr 19) |
| 19. | List any two techniques used for eliciting requirements. Compare the two techniques and list where each is applicable. (Apr 19) |

Requirements Elicitation is one of the most difficult stages of analysis, with numerous communication barriers existing between the analyst and client that make eliciting requirements difficult. Analysts and clients often speak in different general languages, with analysts often being more technical in nature, while clients will often speak more from a business perspective. This makes common understanding difficult. Several other general challenges in requirements elicitation, including conflicting requirements, unspoken or assumed requirements, difficulty in meeting with relevant stakeholders, stakeholder resistance to change, and not enough time set for meeting with all stakeholders.

The requirements elicitation and analysis has 4 main process

We typically start by gathering the requirements, this could be done through a general discussion or interviews with your stakeholders, also it may involve some graphical notation. Then you organize the related requirements into sub components and prioritize them, and finally, you refine them by removing any ambiguous requirements that may arise from some conflicts.

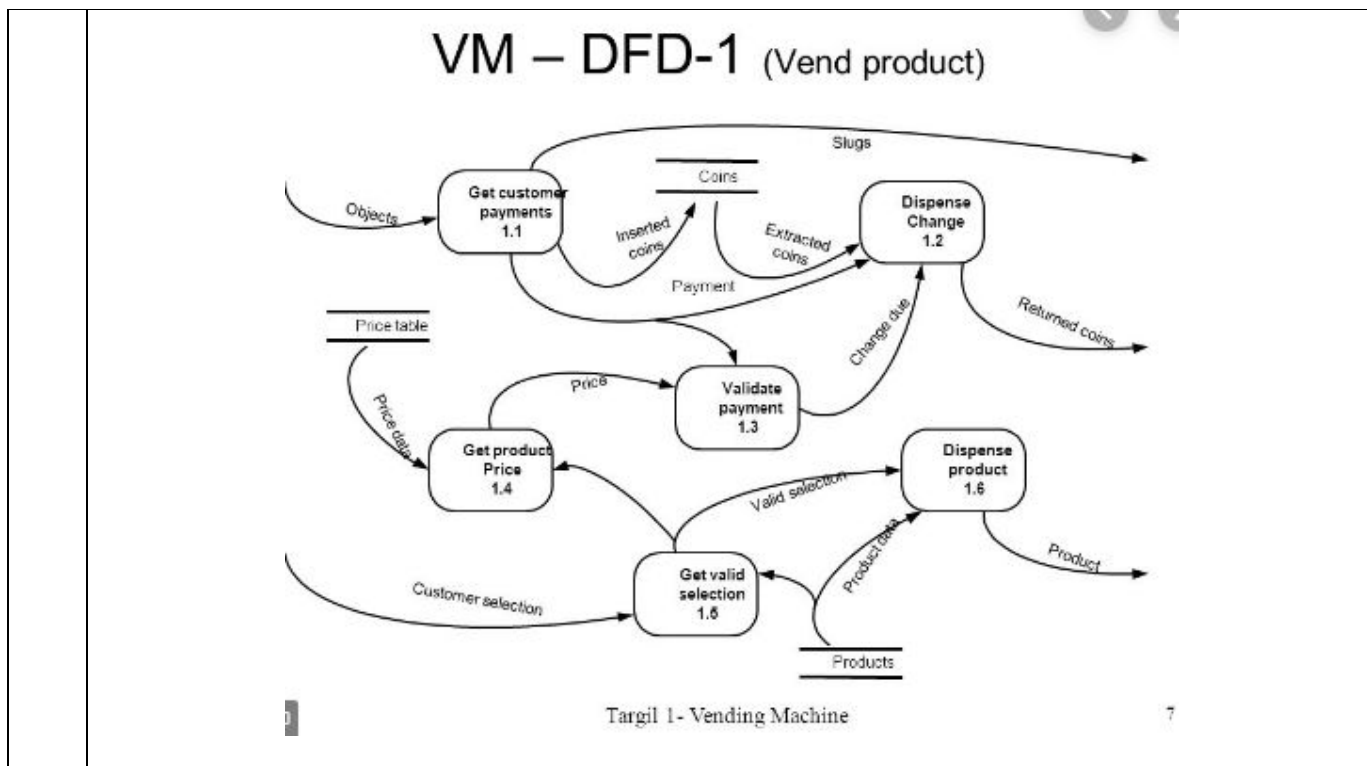
Here are the 4 main process of requirements elicitation and analysis.



The process of requirements elicitation and analysis

20. i) Draw the Level 0 and Level 1 Data Flow diagram for the following system. (8)
 ii) Identify entities in the system and draw a diagram showing the relationship between Entities. (Apr 19)

The Chocolate Vending Machine (CVM) system requirements are as follows: the CVM dispenses chocolates: i) very large chocolates (VC) at Rs. 15, 2) large chocolates (LC) at Rs. 10, and 3) a small chocolates (SC) at Rs. 5. The vending machine only deals in coins. The CVM gives the proper change after the product selection is made. The CVM must check the amount deposited by the customer. The vending machine operates in the following way. A) the CVM remain idle until a customer or owner begins to interact with the machine. When a selection button is pressed the VM indicated the require amount (Rs. 15/Rs. 10/Rs. 5) B) if the full amount needed has been deposited then dispense the proper chocolate and display : Thank You! C) if an insufficient amount (possibly zero) has been deposited display : remaining amount needed. D) if an over amount has need deposited then dispense the proper candy and change and display : Thank You!



UNIT III SOFTWARE DESIGN

Design process – Design Concepts-Design Model- Design Heuristic – Architectural Design -Architectural styles, Architectural Design, Architectural Mapping using Data Flow- User Interface Design: Interface analysis, Interface Design –Component level Design: Designing Class based components, traditional Components.

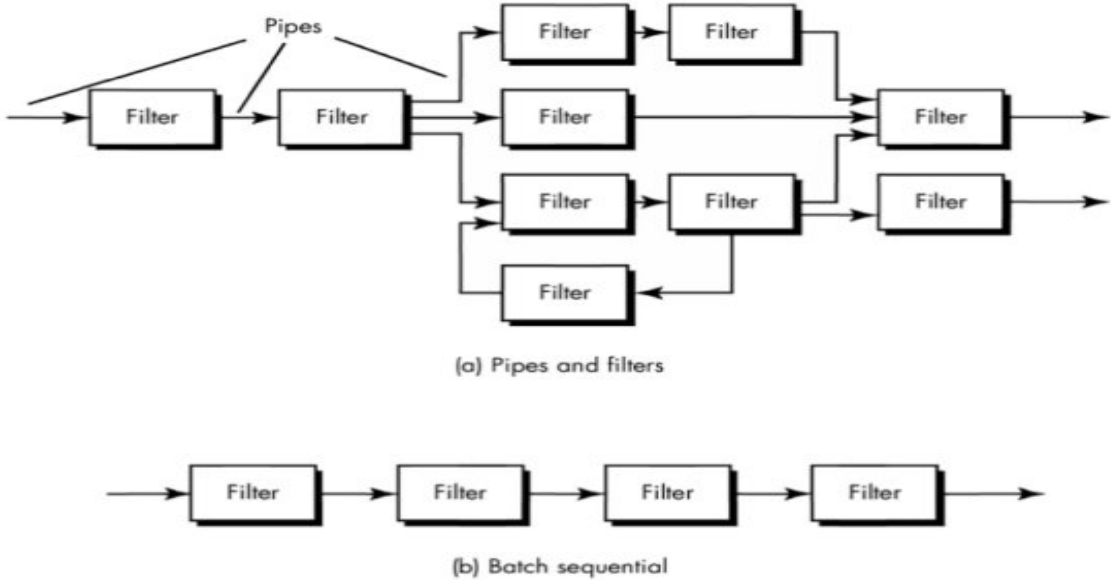
UNIT- III/PART -A

1. **Define information hiding and modularity.(Nov 14)**
 The principle of information hiding suggests that modules be “characterized by design decisions that hides from all others”. In other words, modules should be specified and designed so that information contained within a module is inaccessible to other modules.
 “Modularity is a single attribute of software that allows a program to be intellectually manageable”.
 Monolithic software cannot be easily grasped by a software engineer. This leads to “divide and conquer” strategy – it’s easier to solve a complex problem when you break it into manageable pieces which would reduce the effort.
2. **Define design process.**
 Design process is a sequence of steps carried through which the requirements are translated into a system or software model.
3. **List the architectural models that can be developed. (Nov 10)**
 Architectural models that may be developed may include:
 - **A static structural model** shows the sub-system or components that are to be developed as separate units.
 - **A dynamic process model** shows how the system is organized into processes at run-time.
 - **An interface model** defines the services offered by each sub-system through its public interface.
 - **Relationship model** shows relationships, such as data flow between the subsystems.
 - **A distribution model** shows how sub-systems may be distributed across computers
4. **What is a pattern?**

	A pattern is an insight which conveys the essence of a proven solution to a recurring problem within a certain context amidst competing concerns.	
5.	What are the characteristics of External design? External design of software design involves conceiving, planning out and specifying the external observable characteristics of the software product.	
6.	Distinguish fan-in and fan-out. (Nov 11)	
	FAN-OUT	FAN-IN
	Fan-out is a measure of the number of modules that are directly controlled by another module.	Fan-in indicates how many modules directly control a given modules.
7.	Define procedure and data abstraction. A procedural abstraction is a named sequence of instructions that has a specific and limited function. A data abstraction is named collection of data that describes a data object.	
8.	What is the work product of software design process and who does this? A design model that encompasses architectural, interface, component level and their representations is the primary work product that is produced during software design. Software engineers conduct each of the design tasks.	
9.	List out design methods. (Apr 12) The design methods can be listed as: <ul style="list-style-type: none"> • Data abstraction • System architecture • Design patterns • Modularity • Information hiding • Functional independence • Refinement • Refactoring and Design classes 	
10.	Define Cohesion and Coupling. Cohesion implies that the component or class encapsulates only attributes and operations that are closely related to one another and to the class or component itself. Coupling is a quantitative measure of the degree to which classes are connected to one another. As classes become more interdependent, coupling increases.	
11.	What is the use of Architectural design? The Architectural design defines the relationship between major structural elements of the software, the “design patterns” that can be used and the constraints that affect the way in which architectural design patterns can be applied.	
12.	Explain the qualitative criteria for measuring independence. (Nov 11) Independence is assessed using two qualitative criteria: Cohesion is an indication of the relative functional strength of a module. Coupling is the indication of the relative interdependence among modules.	
13.	Define software architecture and mention its characteristics. Software architecture alludes to “the overall structure of the software and the ways in which that structure provides conceptual integrity for a system”. Structural properties, Extra functional properties, Families of related systems.	
14.	What are the three distinct types of activities of software design? External design, Architectural design and Detailed design.	
15.	Which is referred as internal design? Mention its characteristics.	

	Architectural and detailed designs are collectively referred to as internal design. Internal design involves conceiving, planning out and specifying the internal structure and processing details of the software product.
16.	Define super-ordinate and subordinate. The control relationship among modules is expressed in the following way: A module that controls another module is said to be super ordinate to it and conversely, a module controlled by another is said to be subordinate to the controller.
17.	What are the goals of internal design? The goals of Internal design are to specify internal design and processing details, to record design decisions, to elaborate the test plan and to provide a blueprint for implementation testing and maintenance activities.
18.	What are the work products of internal design? The products of internal design include a specification of architectural structure, the details of algorithms and data structures and the test plan.
19.	What are the fundamental concepts of software design? Fundamental concepts of software design include abstraction, structure information hiding, modularity, concurrency, and verification and design aesthetics.
20.	Define abstraction. Abstraction is the intellectual tool that allows dealing with concept apart from particular instances of those concepts. During requirements definition and design, abstraction permits separation of the conceptual aspects of a system from the implementation details.
21.	Define Archetypes. An archetype is a class or pattern that represents a core abstraction that is critical to the design of the architecture for the target system.
22.	List four design principles of a good design. (Apr 11, Nov 13) The principles of a good design are: <ul style="list-style-type: none"> • The design must implement all of the explicit requirements contained in the analysis model. • It must accommodate all of the implicit requirements desired by the customer. • The design must be readable, understandable guide for those who generate code and for those who test and subsequently support the software. The design should provide a complete picture of the software, addressing the data, functional, and behavioral domains from an implementation perspective.
23.	What are the steps for interface design? (Apr 15) <ul style="list-style-type: none"> • Using information developed during interface analysis, the interface objects and operations are defined. • The events that will cause the change in the state of the user interface are defined and modeled. • Each interface state is depicted as it will actually look to the end-user. An indication of how the user interprets the state of the system from information provided through the interface is provided.
24.	If a module has logical cohesion, what kind of coupling is this module likely to have? (Apr 16) When a module that performs a tasks that are logically related with each other is called logically cohesive. For such module CONTENT COUPLING can be suitable for coupling with another modules. The content coupling is a coupling when one module makes use of data or control information maintained in another module.
25.	What are the principles of Component-level design? <ul style="list-style-type: none"> • The Open-Closed Principle(OCP)

	<ul style="list-style-type: none"> • The Liskov-Substitution Principle(LSP) • Dependency Inversion Principle(DIP) • The Interface Segregation Principle(ISP) • The Release Reuse Equivalency Principle(REP) • The Common Closure Principle(CRP)
26.	<p>What is the need for architectural mapping using data flow? (Apr 16)</p> <ul style="list-style-type: none"> • Mainly for Transform Mapping • Review the fundamental system model. • Review and refine data flow diagrams for the software • Determine whether the DFD has transform or transaction flow characteristics. • Isolate the transform center by specifying incoming and outgoing flow boundaries. • Perform “first-level factoring” • Perform “second-level factoring” <p>Refine the first-iteration architecture using design heuristics for improved software quality.</p>
27.	<p>What UI design patterns are used for the following? (Nov 16, Apr 17)</p> <ol style="list-style-type: none"> Page layout –Layout UI design patterns Tables – Table design pattern Navigation through menus and web pages – Graphical User Interface Shopping cart-Miscellaneous UI design patterns or Web UI design patterns
28.	<p>Draw the context flow graph of ATM automation system. (Nov 17)</p>
29.	<p>Draw diagrams to demonstrate architectural styles.(Apr 15)</p> <p>Data-Centric Architecture</p> <p>Data-Flow Architecture</p>

	 <p>(a) Pipes and filters</p> <p>(b) Batch sequential</p>
30.	<p>Write a note on FURPS model. (Nov 17)</p> <p>Functionality is assessed by evaluating the feature set and capabilities of the program, the generality of the functions that are delivered, and the security of the overall system.</p> <p>Usability is assessed by considering human factors , overall aesthetics, consistency, and documentation.</p> <p>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.</p> <p>Performance is measured by processing speed, response time, resource consumption, throughput, and efficiency.</p> <p>Supportability combines the ability to extend the program (extensibility), adaptability, serviceability – these three attributes represent a more common term, maintainability – in addition, testability, compatibility, configurability , the ease with which a system can be installed, and the ease with which problems can be localized.</p>
31.	<p>What architectural styles are preferred for the following systems? Why? a)networking b)web based systems c)Banking Systems. (Nov 16)</p> <p>a)Networking – Data Flow Architecture</p> <p>Definition This architecture is applied when input data are to be transformed through a series of computational or manipulative components into output data.</p> <p>Reason Networking allows computers to exchange data.</p> <p>b)Web based systems-Data centered Architecture</p> <p>Definition A data store (e.g., a file or database) resides at the center of this architecture and is accessed frequently by other components that update, add, delete, or otherwise modify data within the store.Client software accesses a central repository.</p> <p>Reason</p>

	<p>Web-based applications often run inside a Web browser by sending requests. Web browser publishes and maintains data.</p> <p>c)Banking system-Object oriented Architecture</p> <p>Definition</p> <p>The components of a system encapsulate data and the operations that must be applied to manipulate the data.</p> <p>Reason</p> <p>Banking systems are responsible for operating functionalities such as a payment system, providing loans, taking deposits and helping with investments.</p>						
32.	<p>List the principles of a software design. (Apr 18)</p> <ul style="list-style-type: none"> • The design process should not suffer from “tunnel vision”. • The design should be traceable to the analysis model. • The design should exhibit uniformity and integration. • Design is not coding. The design should not reinvent the wheel. 						
33.	<p>Mention the design quality model proposed by hewlett packard. (Nov 18)</p> <p>a) Functionality b) Usability c) Reliability</p> <p>d) Performance e) Supportability</p>						
34.	<p>Draw the zero level data flow diagram for ATM system. (Nov 18)</p> <pre> graph TD CK[Customer Keypad] --> ATM((ATM)) CS[Control System] --> ATM CR[Card Reader] --> ATM ATM --> AD[Accounts Database] ATM --> S[Screen] ATM --> PD[Printout Dispenser] ATM --> CD[Cash Dispenser] </pre> <p>LEVEL 0 DFD</p>						
35.	<p>What UI design patterns are used for the following? (Apr 18)</p> <ul style="list-style-type: none"> • Page layout – card stack • Tables – sortable table • Navigation through menus and web pages – edit- in-place • Shopping cart – shopping cart 						
36.	<p>Differentiate internal and external design. (Apr 21)</p> <p>The external design is located on the outside of something else, whereas anything that is internal is located on the inside of something and does not involve any input from the outside.</p>						
37.	<p>List out the various types of cohesion and coupling. (Apr 21)</p> <table border="1"> <thead> <tr> <th>Coupling</th><th>Cohesion</th></tr> </thead> <tbody> <tr> <td>Coupling shows the relationships between modules.</td><td>Cohesion shows the relationship within the module.</td></tr> <tr> <td>Coupling shows the relative independence between the modules.</td><td>Cohesion shows the module's relative functional strength.</td></tr> </tbody> </table>	Coupling	Cohesion	Coupling shows the relationships between modules.	Cohesion shows the relationship within the module.	Coupling shows the relative independence between the modules.	Cohesion shows the module's relative functional strength.
Coupling	Cohesion						
Coupling shows the relationships between modules.	Cohesion shows the relationship within the module.						
Coupling shows the relative independence between the modules.	Cohesion shows the module's relative functional strength.						

UNIT-III/ PART- B

1. What is system modeling? Explain the process of creating models and the factors that should be considered when building models. *(Nov 13)*

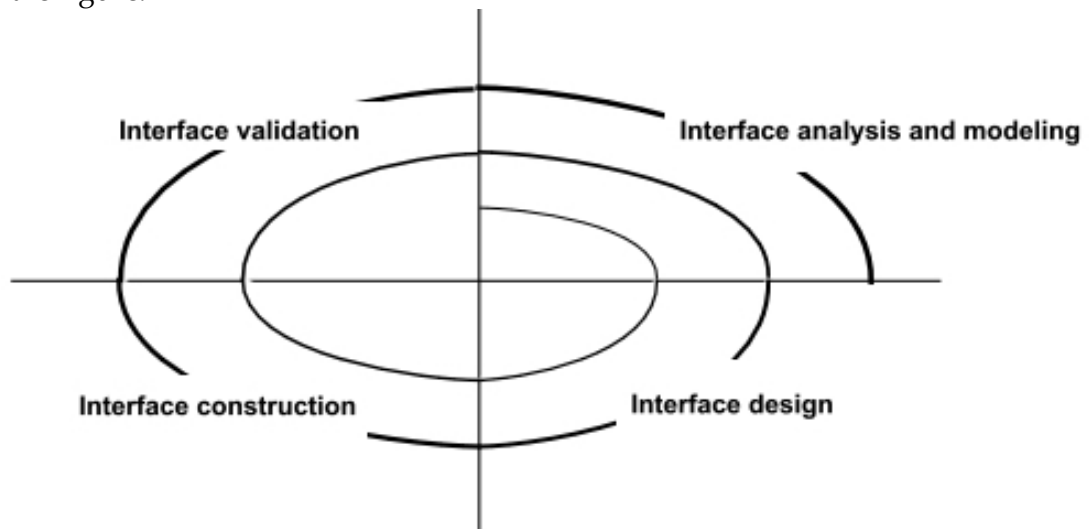
System modelling helps the analyst to understand the functionality of the system and models are used to communicate with customers

- Identify the process necessary to carry out the task you are creating the business process model for.
- Outline the processes necessary in the sequence they need to be performed in order to accomplish the business tasks presented by the process model.
- Discuss and document the business process in which you are creating the process model for.
- Create a flow chart of the steps necessary to achieve the process in which you are creating the model for.
- Begin the process by writing the first step to your task inside of an oval shape.
- Write the next step as an activity performed in the process using a rectangle shape
- Write yes or no questions in diamond shapes for steps that require decisions
- Create more rectangle boxes after your decision diamond in order to show what active tasks are necessary to continue the task's process.
- Finish the flow chart just as you started it, by using an oval shape to signify it's completion.

Go over your flow chart numerous times with both employees who know the process as well as employees who have never done the process in order to ensure it is complete and successfully accuracy.

2. Explain the core activities involved in user interface design process with necessary block diagrams. *(Nov 10, 13, 17, 18 , Apr 21)*

User interface (UI) design is an iterative process where users interact with designers and interface prototypes to decide on the features, organization and the look and feel of the system user interface. Sometimes the interface is separately prototyped in parallel with other software engineering activities. When iterative development is used, the user interface design proceeds incrementally as the software is developed. In both cases before programming starts, some paper-based designs must be developed and ideally tested. The overall UI design process is shown in the figure.



THE USER INTERFACE DESIGN PROCESS

There are three core activities in this process:

User analysis: In the user analysis process, develop an understanding of the tasks that the user does, their working environment, the other systems that they use, how they interact with other people in their work. For products with a diverse range of users, develop this understanding through focus groups, trails with potential users and similar exercises.

System prototyping: User interface design and development is an iterative process. Although users may talk about the facilities they need from an interface, it is very difficult for them to be specific until something tangible is seen. So prototype systems are developed and exposed to the users, which can guide the evolution of the interface.

Interface evaluation: Even though there are discussions with the users during the prototype process, a more formalized evaluation activity is required, where information about users' actual experience with the interface is collected. The scheduling of the UI design within the software process depends to some extent on other activities. Prototyping may be used as a part of requirements engineering process and also to start the UI design process at this stage. In iterative processes UI design is integrated with the software development. Like the software itself the UI design may have to be refactored and redesigned during development.

User analysis: A critical user interface design activity is the analysis of the user activities that are to be supported by the computer system. To develop these understanding, task analysis, ethnographic studies, user interviews and observations or a mixture of all methods can be used.

The challenge for engineers involved in user analysis is to find a way to describe user analyses so that they communicate the essence of the tasks to other designers and to the users themselves. UML sequence charts can be used but they can be too technical for the users, so a natural language scenario to represent user activities must be developed.

Cannot expect users' analysis to generate very specific user interface requirements. The analysis helps to understand the needs and concerns of the system users. As more information is obtained how they work, their concerns and their constraints it can be taken into account of the design.

User interface prototyping:

Because of the dynamic nature of user interfaces, textual description and diagrams are not good enough for expressing user interface requirements. Evolutionary or exploratory prototyping with end-user involvement is the only practical way to design and develop graphical user interfaces for software systems. The aim of prototyping is to gain direct experience with the interface. It is difficult to think abstractly about a user interface to explain exactly what is required. But when presented with examples it is easy to identify the characteristics that are liked and disliked.

When prototyping a user interface, a two-stage prototyping process is adopted:

- Very early in the process, develop paper prototypes-mock ups of screen designs-and walk through these with end-users.
- Then refine the design and develop increasingly sophisticated automated prototypes,
- then make them available to users for testing and activity simulation.

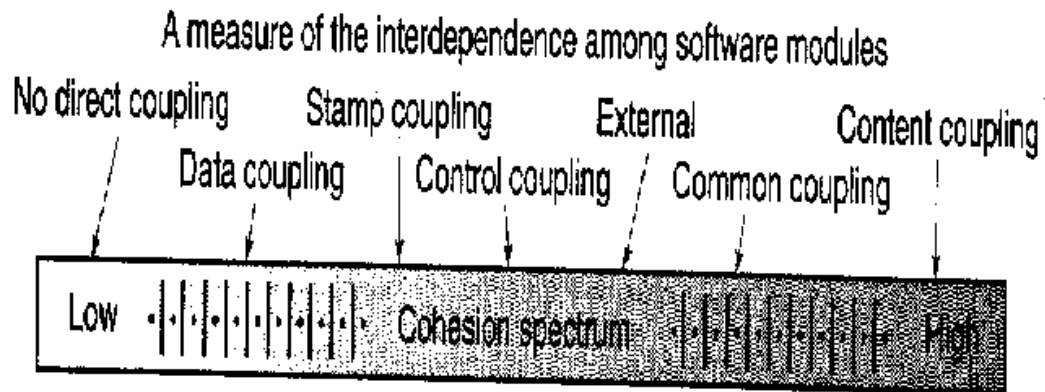
There are three approaches that can be used for user interface prototyping:

- Script-driven approach
- Visual programming languages
- Internet-based prototyping

Interface evaluation:

Interface evaluation is the process of assessing the usability of an interface and checking that it meets user requirements. It should be part of the normal verification and validation process for software systems.

3. (i) Discuss the design heuristics for effective modularity design. (Nov 16)
 (ii) Explain the architectural styles used in the architectural design. (Apr 13, 14, 17)
- Coupling describes the interconnection among modules



- Data coupling
 - Occurs when one module passes local data values to another as parameters
- Stamp coupling
 - Occurs when part of a data structure is passed to another module as a parameter
- Control Coupling
 - Occurs when control parameters are passed between modules
- Common Coupling
 - Occurs when multiple modules access common data areas such as Fortran Common or C extern
- Content Coupling
 - Occurs when a module data in another module
- Subclass Coupling
 - The coupling that a class has with its parent class

DESIGN HEURISTICS

- Evaluate 1st iteration to reduce coupling & improve cohesion
- Minimize structures with high fan-out; strive for depth
- Keep scope of effect of a module within scope of control of that module
- Evaluate interfaces to reduce complexity and improve consistency
- Define modules with predictable function & avoid being overly restrictive
- Avoid static memory between calls where possible
- Strive for controlled entry -- no jumps into the middle of things
- Package software based on design constraints and portability requirements

- (ii) Explain the architectural styles used in the architectural design. (MAY/JUNE 2013, 2014)

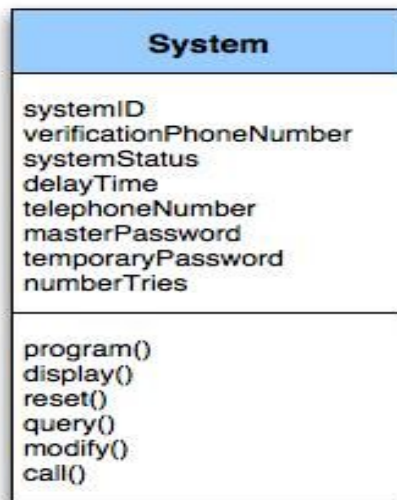
ARCHITECTURAL DESIGN

- Establishing the overall structure of a software system
- Objectives
 - To introduce architectural design and to discuss its importance
 - To explain why multiple models are required to document a software architecture
 - To describe types of architectural model that may be used
- Architectural (high-level) design = the process of establishing the subsystems of a larger software system and defining a framework for subsystem control and communication
- Software architecture = the output of the high-level design process
- Defining and documenting the software architecture provides support for:

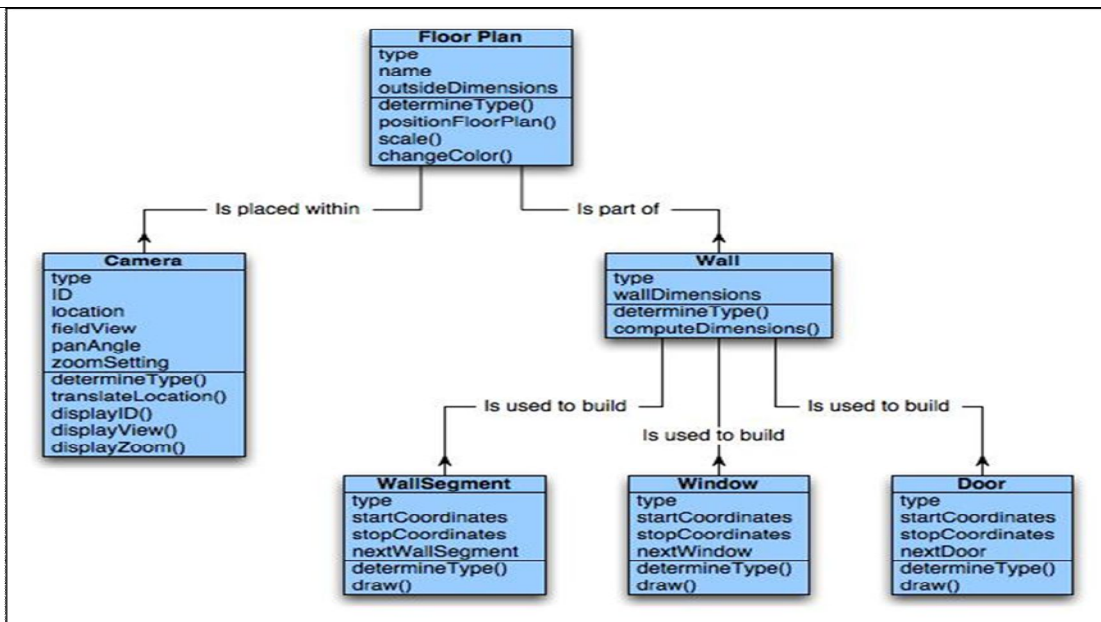
	<ul style="list-style-type: none"> – Stakeholder communication – System analysis – Large-scale software reuse <ul style="list-style-type: none"> • The software architecture of a program or computing system is the structure or structures of the system, which comprise software components • architecture is not the operational software • Enables a software engineer <ul style="list-style-type: none"> ➤ Analyze the effectiveness of the design in meeting its stated requirements ➤ Consider architectural alternatives at a stage when making design changes is still relatively easy ➤ Reducing the risks associated with the construction of the software. <ul style="list-style-type: none"> • Representations of software architecture are an enabler for communication between all parties • The architecture highlights early design decisions that will have a profound impact on all software engineering work • Architecture “constitutes a relatively small, intellectually graspable model of how the system is structured <p>Architectural Styles</p> <ul style="list-style-type: none"> ➤ Each style describes a system category that encompasses: <ol style="list-style-type: none"> 1. a set of components (e.g., a database, computational modules) that perform a function required by a system, 2. a set of connectors that enable “communication, coordination, and cooperation” among components, 3. constraints that define how components can be integrated to form the system, and 4. semantic models that enable a designer to understand the overall properties of a system. <p>Specific Styles</p> <ul style="list-style-type: none"> • Data-centered architecture • Data flow architecture • Call and return architecture • Object-oriented architecture • Layered architecture
4.	<p>Discuss class-based components along with its principles in detail. For a Case study of your choice show the architectural and Component design. (Apr 15)</p> <p>Identifying Analysis Classes</p> <ul style="list-style-type: none"> ➤ External entities that produce or consume information ➤ Things that are part of the information domain ➤ Occurrences or events ➤ Roles played by people who interact with the system ➤ Organizational units ➤ Places that establish context ➤ Structures that define a class of objects <p>Class Selection Criteria</p> <ul style="list-style-type: none"> • Retained information • Needed services • Multiple attributes • Common attributes • Common operations • Essential requirement

Identifying Classes

Potential class	Classification	Accept / Reject
homeowner	role; external entity	reject: 1, 2 fail
sensor	external entity	accept
control panel	external entity	accept
installation	occurrence	reject
(security) system	thing	accept
number, type	not objects, attributes	reject: 3 fails
master password	thing	reject: 3 fails
telephone number	thing	reject: 3 fails
sensor event	occurrence	accept
audible alarm	external entity	accept: 1 fails
monitoring service	organizational unit; ee	reject: 1, 2 fail

Class Diagram

Class diagram for the system class



Class Responsibilities

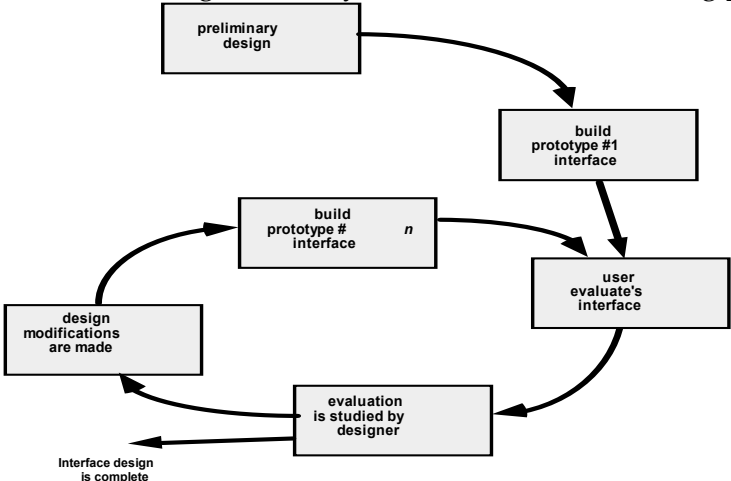
- Distribute system intelligence across classes.
- State each responsibility as generally as possible.
- Put information and the behavior related to it in the same class.
- Localize information about one thing rather than distributing it across multiple classes.
- Share responsibilities among related classes, when appropriate.

Class Collaborations

➤ Relationships between classes:

- is-part-of — used when classes are part of an aggregate class.
- has-knowledge-of — used when one class must acquire information from another class.
- depends-on — used in all other cases.

5. Explain the interface analysis and discuss the goals of task analysis involved in detail.
 - Interface analysis means understanding
 - (1) the people (end-users) who will interact with the system through the interface;
 - (2) the tasks that end-users must perform to do their work,
 - (3) the content that is presented as part of the interface
 - (4) the environment in which these tasks will be conducted.
 - Are users trained professionals, technician, clerical, or manufacturing workers?
 - What level of formal education does the average user have?
 - Are the users capable of learning from written materials or have they expressed a desire for classroom training?
 - Are users expert typists or keyboard phobic?
 - What is the age range of the user community?
 - Will the users be represented predominately by one gender?
 - How are users compensated for the work they perform?
 - Do users work normal office hours or do they work until the job is done?
 - Is the software to be an integral part of the work users do or will it be used only occasionally?
 - What is the primary spoken language among users?
 - What are the consequences if a user makes a mistake using the system?
 - Are users experts in the subject matter that is addressed by the system?

	<ul style="list-style-type: none"> Do users want to know about the technology the sits behind the interface? <p>Task Analysis and Modeling</p> <ul style="list-style-type: none"> Task Analysis answers the following questions ... <ul style="list-style-type: none"> What work will the user perform in specific circumstances? What tasks and subtasks will be performed as the user does the work? What specific problem domain objects will the user manipulate as work is performed? What is the sequence of work tasks – the workflow? What is the hierarchy of tasks? Use-cases define basic interaction Task elaboration refines interactive tasks Object elaboration identifies interface objects (classes) Workflow analysis defines how a work process is completed when several people (and roles) are involved
6.	<p>What are the issues that have to be constrained in the design process? Explain the design evaluation in detail. Once the first prototype is built, the designer can collect a variety of qualitative and quantitative data that will assess in evaluating the interface. Questions can be a simple Y/N response, numeric response, scaled response, Likert scale (strongly agree, etc.), percentage response, and open-ended ones.</p> <p>Design Issues</p> <ul style="list-style-type: none"> Response time: System response time has 2 important characteristics: length and variability. Variability refers to the deviation from average response time. Help facilities: Help must be available for all system functions. Include help menus, print documents. Error handling: describe the problem in a language the user can understand. Never blame the user for the error that occurred. Menu and command labeling: menu options should have corresponding commands. Use control sequences for commands. Application accessibility: especially for the physically challenged. Internationalization: The Unicode standard has been developed to address the daunting challenge of managing dozens of natural languages with hundred of characters and symbols. <p>Design Evaluation Cycle</p> <p>Two interface design evaluation techniques are mentioned in this section, usability questionnaires and usability testing. The process of learning how to design good user interfaces often begins with learning to identify the weaknesses in existing products.</p>  <pre> graph TD A[preliminary design] --> B[build prototype #1 interface] B --> C[user evaluate's interface] C --> D[evaluation is studied by designer] D --> E[design modifications are made] E --> F[build prototype # n interface] F --> C D --> G[Interface design is complete] </pre>

Once the first prototype is built, the designer can collect a variety of qualitative and quantitative data that will assess in evaluating the interface. Questions can be a simple Y/N response, numeric response, scaled response, Likert scale (strongly agree, etc.), percentage response, and open-ended ones.

7. Discuss about software architecture design, which emphasize on fan in, fan out, coupling, cohesion and factoring? Explain in detail types of cohesion and coupling with examples (*Nov 11, 12, Apr 15, 17, 18*)

1. cohesion and factoring? (NOV/DEC 2011) (NOV/DEC 2012)(APR/MAY 2015)

In the AD phase, the software requirements are transformed into definitions of software components and their interfaces, to establish the framework of the software. This is done by examining the SRD and building a 'physical model' using recognized software engineering methods. The physical model should describe the solution in concrete, implementation terms. Just as the logical model produced in the SR phase structures the problem and makes it manageable, the physical model does the same for the solution.

The physical model is used to produce a structured set of component specifications that are consistent, coherent and complete. Each specification defines the functions, inputs and outputs of the component

The major software components are documented in the Architectural Design Document (ADD). The ADD gives the developer's solution to the problem stated in the SRD. The ADD must cover all the

requirements stated in the SRD (AD20), but avoid the detailed consideration of software requirements that do not affect the structure.

The main outputs of the AD phase are the:

- Architectural Design Document (ADD);
- Software Project Management Plan for the DD phase (SPMP/DD);
- Software Configuration Management Plan for the DD phase(SCMP/DD);
- Software Verification and Validation Plan for the DD Phase (SVVP/DD);
- Software Quality Assurance Plan for the DD phase (SQAP/DD);
- Integration Test Plan (SVVP/IT).

Figure 2.1 summarizes activities and document flow in the AD phase. The following subsections describe the activities of the AD phase in more detail.

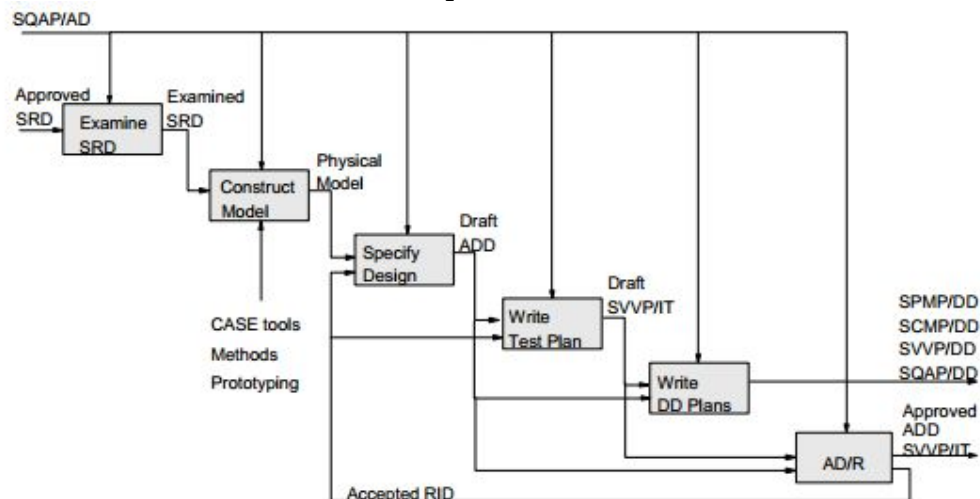


Figure 2.1: AD phase activities

Design quality

Designs should be adaptable, efficient and understandable. Adaptable designs are easy to modify and maintain. Efficient designs make minimal use of available resources. Designs must be understandable if they are to be built, operated and maintained effectively.

Attaining these goals is assisted by aiming for simplicity. Metrics should be used for measuring simplicity/complexity (e.g. number of interfaces per component and cyclomatic complexity).

Simple components are made by maximizing the degree to which the activities internal to the component are related to one another (i.e. 'cohesion').

Simple structures can be made by:

- minimizing the number of distinct items that are passed between components (i.e. 'coupling');
- ensuring that the function a component performs is appropriate to its place in the hierarchy (i.e. 'system shape');
- ensuring the structure of the software follows the structure of the data it deals with (i.e. 'data structure match');
- designing components so that their use by other components can be maximised (i.e. 'fan-in');
- restricting the number of child components to seven or less (i.e. 'fanout');
- removing duplication between components by making new components (i.e. 'factoring').

Designs should be 'modular', with minimal coupling between components and maximum cohesion within each component. Modular designs are more understandable, reusable and maintainable than designs consisting of components with complicated interfaces and poorly matched groupings of functions. Each level of a design should include only the essential aspects and omit inessential detail. This means they should have the appropriate degree of 'abstraction'. This enhances understandability, adaptability and maintainability.

Design metrics**1. Cohesion**

The parts of a component should all relate to a single purpose.

Cohesion measures the degree to which activities within a component are related to one another. The cohesion of each component should be maximized. High cohesion leads to robust, adaptable, maintainable software.

Cohesion should be evaluated externally (does the name signify a specific purpose?) and internally (does it contain unrelated pieces of logic?).

Types of cohesion:

- functional cohesion;
- sequential cohesion;
- communicational cohesion;
- procedural cohesion;
- temporal cohesion;
- logical cohesion;
- coincidental cohesion.

Functional cohesion is most desirable and coincidental cohesion should definitely be avoided. The other types of cohesion are acceptable under certain conditions. The seven types are discussed below with reference to typical components that might be found in a statistics package.

Functionally cohesive components contain elements that all contribute to the execution of one and only one problem-related task. Names carry a clear idea of purpose (e.g. CALCULATE_MEAN). Functionally cohesive components should be easy to reuse. All the components at the

bottom of a structure should be functionally cohesive.

Sequentially cohesive components contain chains of activities, with the output of an activity being the input to the next. Names consist of specific verbs and composite objects (e.g. CALCULATE_TOTAL_AND_MEAN). Sequentially cohesive components should be split into functionally cohesive components.

Communicational cohesive components contain activities that share the same input data. Names have over-general verbs but specific objects (e.g. ANALYSE_CENSUS_DATA), showing that the object is used for a range of activities. Communicational cohesive components should be split into functionally cohesive components.

Procedurally cohesive components contain chains of activities, with control passing from one activity to the next. Individual activities are only related to the overall goal rather than to one another. Names of procedurally cohesive components are often 'high level', because they convey an idea of their possible activities, rather than their actual activities (e.g. PROCESS_CENSUS_DATA). Procedurally cohesive components are acceptable at the highest levels of the structure.

Temporally cohesive components contain activities that all take place simultaneously. The names define a stage in the processing (e.g. INITIALISE, TERMINATE). Bundling unrelated activities together on a temporal basis is bad practice because it reduces modularity. An initialization component may have to be related to other components by a global data area, for example, and this increases coupling and reduces the possibilities for reuse. Temporal cohesion should be avoided.

Logically cohesive components perform a range of similar activities. Much of the processing is shared, but special effects are controlled by means of flags or 'control couples'. The names of logically cohesive components can be quite general, but the results can be quite specialized (e.g. PROCESS_STATISTICS). Logical cohesion causes general sounding names to appear at the bottom levels of the structure, and this can make design less understandable. Logical cohesion should be avoided, but this may require significant restructuring. Coincidentally cohesive components contain wholly unrelated activities. Names give no idea of the purpose of the component (e.g. MISCELLANEOUS_ROUTINE). Coincidentally cohesive components are often invented at the end of the design process to contain miscellaneous activities that do not neatly fit with any others. Coincidental cohesion severely reduces understandability and maintainability, and should be avoided, even if the penalty is substantial redesign.

Practical methods for increasing cohesion are to:

- make sure that each component has a single clearly defined function;
- make sure that every part of the component contributes to that function;
- keep the internal processing short and simple.

2. coupling

A 'couple' is an item of information passed between two components. The 'coupling' between two components can be measured by the number of couples passed. The number of couples flowing into and out of a component should be minimized.

The term 'coupling' is also frequently used to describe the relative independence of a component. 'Tightly' coupled components have a high level of interdependence and 'loosely' coupled components have a low level of interdependence. Dependencies between components should be minimized to maximize reusability and maintainability.

five types of coupling:

- data coupling;
- stamp coupling;

	<ul style="list-style-type: none"> • control coupling; • common coupling; • content coupling. <p>Data coupling is most desirable and content coupling should be avoided. The other types of coupling are all acceptable under certain conditions, which are discussed below. Data coupling is passing only the data needed. The couple name is a noun or noun phrase. The couple name can be the object in the component name. For example a component called 'CALCULATE_MEAN' might have an output data couple called 'MEAN'.</p> <p>Stamp coupling is passing more data than is needed. The couple name is a noun or noun phrase. Stamp coupling should be avoided. Components should only be given the data that is relevant to their task. An example of stamp coupling is to pass a data structure called 'CENSUS_DATA' to a component that only requires the ages of the people in the census to calculate the average age.</p> <p>Control coupling is communication by flags. The couple name may contain a verb. It is bad practice to use control coupling for the primary control flow; this should be implied by the component hierarchy. It is acceptable to use control coupling for secondary control flows such as in error handling.</p> <p>Common coupling is communication by means of global data, and causes data to be less secure because faults in components that have access to the data area may corrupt it. Common coupling should normally be avoided, although it is sometimes necessary to provide access to data by components that do not have a direct call connection (e.g. if A calls B calls C, and A has to pass data to C).</p> <p>Content coupling is communication by changing the instructions and internal data in another component. Communication coupling is bad because it prevents information hiding. Only older languages and assembler allow this type of coupling.</p> <p>Practical methods of reducing coupling are to:</p> <ul style="list-style-type: none"> ▪ structure the data; ▪ avoid using flags or semaphores for primary control flow and use messages or component calls; ▪ avoid passing data through components that do not use it; ▪ provide access to global or shared data by dedicated components. ▪ make sure that each component has a single entry point and exit point. <p>Fan-in The number of components that call a component measures its 'fan-in'. Reusability increases as fan-in increases. Simple fan-in rules are: 'maximize fan-in' and 'reuse components as often as possible'.</p> <p>Fan-out The number of components that a component calls measures its 'fan-out'. Fan-out should usually be small, not more than seven. However some kinds of constructs force much higher fan-out values (e.g. case statements) and this is often acceptable. Simple fan-out rules are: 'make the average fan-out seven or less' or 'make components depend on as few others as possible'.</p>
8.	<p>Explain the basic concepts of software design. (<i>Apr 11 , 19 , Nov 14</i>)</p> <p>A set of fundamental software design concepts has evolved over the history of software engineering. M. A. Jackson once said: "The beginning of wisdom for a [software engineer] is to recognize the difference between getting a program to work, and getting it right."</p> <p>Abstraction</p>

When we consider a modular solution to any problem, many levels of abstraction can be posed. At the highest level of abstraction, a solution is stated in broad terms using the language of the problem environment. At the lower levels of abstraction, a more detailed description of the solution is provided.

A procedural abstraction refers to a sequence of instructions that have a specific and limited function. And on this abstraction details are suppressed. An example of a procedural abstraction would be the word open for a door. Open implies a long sequence of procedural steps (e.g., walk to the door, reach out and grasp knob, turn knob and pull door, step away from moving door, etc.).

A data abstraction is a named collection of data that describes a data object. In the context of the procedural abstraction open, we define a data abstraction called **door**. Like any data object, the data abstraction for **door** would encompass the a set of attributes that describe the door (e.g., door type, swing direction, opening mechanism, weight, dimensions). It follows that the procedural abstraction open would make use of information contained in the attributes of the data abstraction **door**.

Architecture

Software architecture alludes to “the overall structure of the software and the ways in which that structure provides conceptual integrity for a system”. Architecture is the structure or organization of program components (modules), the manner in which these components interact , and the structure of data that are used by the components. 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 programs of program components. Framework models increase the level of design abstraction by attempting to identify repeatable architectural design frameworks that are encountered in similar types of application. Dynamic models address the behaviour 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 the business or technical process that the system must accommodate. Finally, functional models can be used to represent the functional hierarchy of a system.

Patterns

“A pattern is a named nugget of insight which conveys the essence of a proven solution to a recurring problem within a certain context amidst competing 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 functionality or structurally different pattern.

Modularity

Software architecture and design patterns embody modularity. “ Modularity is a single attribute of software that allows a program to be intellectually manageable”. Monolithic software cannot be easily grasped by a software engineer. Consider two problems, p1 and p2. If the perceived complexity of p1 is greater than the perceived complexity of p2, it follows that the effort required to solve p1 is greater than the effort required to solve p2. As a general case, this result is intuitively obvious. It takes more time to solve a difficult problem. It also follows that the perceived complexity of two problems when they are combined is often greater than the sum of the perceived complexity when each is taken separately. This leads to “divide and conquer” strategy – it’s easier to solve a complex problem when you break it into manageable pieces. This has important implications with regard to modularity

	<p>and software. If we subdivide software indefinitely, the effort required to develop it will become negligibly small.</p> <p>Information hiding</p> <p>The principle of information hiding suggest that modules be “characterized by design decisions that hides from all others.” Modules should be specified and designed so that information contained within a module is inaccessible to other modules that have no need for such information.</p> <p>Functional Independence</p> <p>The concept of functional independence is a direct outgrowth of modularity and the concepts of abstraction and information hiding. It is achieved by developing modules with “single minded” function and an “aversion” to exercise interaction with other modules. Independence is assessed using two qualitative criteria: cohesion and coupling. Cohesion is an indication of the relative functional strength of a module. Coupling is a indication of the relative interdependence among modules.</p> <p>Refinement</p> <p>Stepwise refinement is a top-down design strategy. A program is developed successively refining levels of procedure detail. A hierarchy is developed by decomposing a macroscopic statement of function in a stepwise fashion until programming language statements are reached. Refinement is actually a process of elaboration. Abstraction and refinement are complementary concepts.</p> <p>Refactoring</p> <p>It is the process of a changing a software system in such a way that it does not alter the external behaviour of the code yet proves its internal structure. When software is refactored, then the existing design is examined for redundancy, unused design elements, inefficient or unnecessary algorithms, poorly constructed or inappropriate data structures, or any other design failure that can be corrected to yield a better design.</p> <p>Design Classes</p> <p>As the design model evolves, the software team must define a set of design classes that</p> <ul style="list-style-type: none"> • refine the analysis classes by providing design detail that will enable the classes to be implemented, and • create a new set of design classes that implement a software infrastructure to support the business solution. <p>Five different types of design classes are created by the designer. They are</p> <ul style="list-style-type: none"> • User interface classes • Business domain classes • Process classes • Persistent classes • System classes <p>Design classes can be reviewed to ensure that it is “well-formed”. So four characteristics of a well-formed design classes are</p> <ul style="list-style-type: none"> • Complete and sufficient • Primitiveness • High cohesion • Low Coupling
9.	<p>Explain the various modular decomposition and control styles commonly used in any organizational model. (<i>Nov 10</i>)</p> <p>After a structural architecture has been designed , the next stage of the architectural design process is the decomposition of sub-systems into modules.</p>

	<p>Two models are used when decomposing a sub-system into modules:</p> <p>An object-oriented model</p> <p>The system is decomposed into a set of communicating objects. Modules are objects with private state and defined operations on that state. In the data-flow model, modules are functional transformations.</p> <p>Object Models</p> <p>The system is divided into a set of loosely coupled objects with well-defined interfaces. Objects call on the services provided by other objects. The system can issue invoices to customers, receive payments, issue receipts for these payments and remainders for unpaid invoices. Operations are represented as rounded rectangle representing the object. Dashed arrow indicates that an object uses the attributes or services provided by other object.</p> <p>This decomposition is concerned with object classes, their attribute and operations. When implemented, objects are created from these classes and some control model is used to coordinate object operations.</p> <p>Advantages</p> <ul style="list-style-type: none"> • Since objects are loosely coupled, the implementation of objects can be modified without affecting other objects. • Objects are often represented of real -world entities so the structure of the system is readily understandable. • Objects can be reused. • Object-oriented programming languages have been developed which provide direct implementations of architectural components. <p>Disadvantages</p> <ul style="list-style-type: none"> • To use services, objects must explicitly reference the name and the interface of other objects. • More complex entities are sometimes difficult to represent as objects. <p>Data-flow models</p> <p>In a data-flow model, functional transformations process their inputs and produce output Data flows from one to another and is transformed as it moves through the sequence. Each processing step is implemented as a transform. The transformations may execute sequential or in parallel. The data can be processed by each transform item by item or in a single batch. When the transformations are represented as separate processes, this model is sometime called the pipe and filter model.</p> <p>Advantages</p> <ul style="list-style-type: none"> • It supports the reuse of transformations. • It is intuitive in that many people think of their work in terms of input and output processing. • Evolving the system by adding new transformations is usually straightforward. • It is simple to implement either as a concurrent or a sequential system. <p>Disadvantage</p> <ul style="list-style-type: none"> • Need for a common format for data transfer which can be recognized by all transformations. • Interactive systems are difficult to write using the data-flow model because of the need for a stream of data to be processed.
10.	<p>Discuss the process of translating the analysis model into a software design. (<i>Apr 11</i>)</p> <p>It provides the software designer with a representation of information, function, and behaviour that can be translated to architectural, interface, and component-level designs. Finally, the analysis model and the requirements specification provide the developer and the customer with the means to assess quality once software is built.</p>

Overall objectives and Philosophy

It has three primary objectives.

- To describe what the customer requires.
- To establish a basis for the creation of a software design and
- To define a set of requirements that can be validated once the software is built.

Analysis Rule of Thumb

- The model should focus on requirements that are visible within the problem or business domain.
- Each element of the analysis model should add to an overall understanding of the software requirements and provide insight into the information design, function, and behaviour of the system.
- Delay consideration of infrastructure and other non-functional modules until design.
- Minimize coupling throughout the system.
- Be certain that the analysis model provides value to all stakeholders.
- Keep the model as simple as possible.

Domain Analysis

It is the identification, analysis, and specification of common requirements from a specific application domain, typically for reuse on multiple projects within the application domain. The “specific application domain” can range from avionics to banking, from multimedia video games to software embedded within the medical devices.

Analysis Modelling Approaches

One view of analysis modelling, called structured analysis, considers data and the processes that transform the data as separate entities. Data objects are modelled in a way that defines their attributes and relationships.

A second approach to analysis modelling, called object-oriented analysis, focuses on definition of classes and the manner in which they collaborate with one another to effect customer requirements.

Data Modelling Concepts

Analysis modelling begins with data modelling.

➤ Data Object

A data object is a representation of almost any composite information that must be understood by software. A data object can be an external entity. A data object encapsulates data only. e.g., Car

➤ Data Attribute

Data Attribute define the properties of a data object. They can be used to

- Name an instance of the data object
- Describe the instance, or
- Make reference to another instance in another table.

➤ Relationships

Data objects are connected to one another in different ways. Consider the two data objects, **person** and **car**. A connection is established between **person** and **car** because two objects are related.

For example,

- A person owns a car.
- A person is insured to drive a car.

- The relationship owns and insured to drive define the relevant connections between **person and car.**

➤ **Cardinality and Modality**

Cardinality is the specification of the number of occurrences of one object that can be related to the number of occurrences of another object. It also defines “ the maximum number of object that can participate in a relationship”. The modality of a relationship is 0 if there is no explicit need for the

relationship to occur or the relationship is optional. The modality is 1 if an occurrence of the relationship is mandatory.

➤ **Object Oriented Analysis**

To accomplish this, a number of tasks must occur:

- Basic user requirements must be communicated between the customer and the software engineer.
- Classes must be identified.
- Class hierarchy is defined.
- Object to object relationship should be represented.
- Object behaviour must be modelled.
- Above tasks are reapplied iteratively until the model is complete.

➤ **Scenario based Modelling**

➤ **Writing use cases**

The concept of a use case is relatively easy to understand and to describe a specific usage scenario in straight forward language from the point of view of a defined actor.

- What to write about
- How much to write about it
- How detailed to make a description
- How to organise the description

➤ **Flow Oriented Modelling**

A few simple guidelines can aid immeasurably during derivation of a data flow diagram. The level0 data flow diagram should depict the software as a single bubble .Primary input and output should be carefully noted.

- Refinement should begin by isolating candidate process, data objects and data stores to be represented.
- All arrows and bubbles should be labelled with meaningful names
- Information flow continuity must be maintained from level to level
- One bubble at a time should be refined.

➤ **Class based modelling**

Identifying analysis classes:

Analysis classes manifest themselves in one of the following ways

- External entities
- Things
- Occurrences or events
- Roles
- Organizational event
- Places
- Structures

Behavioral Model

The behavioural model indicates how software will respond to external events or stimuli. To create a model, the analyst must perform the following steps:

- Evaluate all use-cases to fully understand the sequence of interaction within the system.

	<ul style="list-style-type: none"> Identify events that derive the interaction sequence and understand how these events relate to specific classes. Create a sequence for each use-case. Build a state diagram for the system. <p>Review the behaviour model to verify accuracy and consistency.</p>
11.	<p>i) What is modularity? State its importance and explain coupling and cohesion. (<i>Apr 16</i>)</p> <p>ii) Discuss the differences between object oriented and functional oriented design. (<i>Apr 16</i>)</p> <p>Software is divided into separately named and addressable components, often called modules, that are integrated to satisfy problem requirements. "Modularity is the single attribute of software that allows a program to be intellectually manageable" Monolithic software (i.e., a large program composed of a single module) cannot be easily grasped by a reader. The number of control paths, span of reference, number of variables, and overall complexity would make understanding close to impossible. To illustrate this point, consider the following argument based on observations of human problem solving.</p> <p>Let $C(x)$ be a function that defines the perceived complexity of a problem x, and $E(x)$ be a function that defines the effort (in time) required to solve a problem x. For two problems, p_1 and p_2, if</p> $C(p_1) > C(p_2)$ <p>it follows that</p> $E(p_1) > E(p_2)$ <p>As a general case, this result is intuitively obvious. It does take more time to solve a difficult problem.</p> <p>Another interesting characteristic has been uncovered through experimentation in human problem solving. That is,</p> $C(p_1 + p_2) > C(p_1) + C(p_2)$ <p>Expression implies that the perceived complexity of a problem that combines p_1 and p_2 is greater than the perceived complexity when each problem is considered separately. Considering Expression and the condition implied by Expressions, it follows that</p> $E(p_1 + p_2) > E(p_1) + E(p_2)$ <p>This leads to a "divide and conquer" conclusion – it's easier to solve a complex problem when you break it into manageable pieces. The result expressed in Expression has important implications with regard to modularity and software. It is, in fact, an argument for modularity. There is a number, M, of modules that would result in minimum development cost, but we do not have the necessary sophistication to predict M with assurance.</p> <p>Meyer defines five criteria that enable us to evaluate a design method with respect to its ability to define an effective modular system:</p> <p>Modular decomposability. If a design method provides a systematic mechanism for decomposing the problem into sub problems, it will reduce the complexity of the overall problem, thereby achieving an effective modular solution.</p> <p>Modular composability. If a design method enables existing (reusable) design components to be assembled into a new system, it will yield a modular solution that does not reinvent the wheel.</p> <p>Modular understandability. If a module can be understood as a standalone unit (without reference to other modules), it will be easier to build and easier to change.</p> <p>Modular continuity. If small changes to the system requirements result in changes to individual modules, rather than system wide changes, the impact of change-induced side effects will be minimized.</p>

Modular protection. If an aberrant condition occurs within a module and its effects are constrained within that module, the impact of error-induced side effects will be minimized.

A cohesive module performs a single task within a software procedure, requiring little interaction with procedures being performed in other parts of a program.

Stated simply, a cohesive module should (ideally) do just one thing.

Cohesion may be represented as a "spectrum." We always strive for high cohesion, although the mid-range of the spectrum is often acceptable. The scale for cohesion is nonlinear. That is, low-end cohesiveness is much "worse" than middle range, which is nearly as "good" as high-end cohesion. In practice, a designer need not be concerned with categorizing cohesion in a specific module. Rather, the overall concept should be understood and low levels of cohesion should be avoided when modules are designed.

At the low (undesirable) end of the spectrum, we encounter a module that performs a set of tasks that relate to each other loosely, if at all. Such modules are termed coincidentally cohesive. A module that performs tasks that are related logically (e.g., a module that produces all output regardless of type) is logically cohesive. When a module contains tasks that are related by the fact that all must be executed with the same span of time, the module exhibits temporal cohesion.

Coupling is a measure of interconnection among modules in a software structure. Coupling depends on the interface complexity between modules, the point at which entry or reference is made to a module, and what data pass across the interface.

In software design, we strive for lowest possible coupling. Simple connectivity among modules results in software that is easier to understand and less prone to a "ripple effect", caused when errors occur at one location and propagate through a system.

Figure 13.6 provides examples of different types of module coupling. Modules a and d are subordinate to different modules. Each is unrelated and therefore no direct coupling occurs. Module c is subordinate to module a and is accessed via a conventional argument list, through which data are passed. As long as a simple argument list is present (i.e., simple data are passed; a one-to-one correspondence of items exists), low coupling (called data coupling) is exhibited in this portion of structure. A variation of data coupling, called stamp coupling, is found when a portion of a data structure (rather than simple arguments) is passed via a module interface. This occurs between modules b and a.

At moderate levels, coupling is characterized by passage of control between modules.

Control coupling is very common in most software designs and is shown in Figure 13.6 where a "control flag" (a variable that controls decisions in a subordinate or superordinate module) is passed between modules d and e. content coupling, occurs when one module makes use of data or control information maintained within the boundary of another module.

Secondarily, content coupling occurs when branches are made into the middle of a module.

(ii) Discuss the differences between object oriented and functional oriented design. (May/June 2016)

1.FOD: The basic abstractions, which are given to the user, are real world functions.

OOD: The basic abstractions are not the real world functions but are the data abstraction where the real world entities are represented.

2.FOD: Functions are grouped together by which a higher level function is Page on obtained.an eg. of this technique is SA/SD.

OOD: Functions are grouped together on the basis of the data they operate since the classes are associated with their methods.

3.FOD: In this approach the state information is often represented in a centralized shared memory.

	<p>OOD: In this approach the state information is not represented in a centralized memory but is implemented or distributed among the objects of the system.</p> <p>4.FOD approach is mainly used for computation sensitive application, OOD: whereas OOD approach is mainly used for evolving system which mimicks a business process or business case.</p> <p>5. In FOD – we decompose in function/procedure level OOD: – we decompose in class level</p> <p>6. FOD: TOp down Approach OOD: Bottom up approach</p> <p>7. FOD: It views system as Black Box that performs high level function and later decompose it detailed function so to be mapped to modules. OOD: Object-oriented design is the discipline of defining the objects and their interactions to solve a problem that was identified and documented during object-oriented analysis.</p> <p>8. FOD: Begins by considering the use case diagrams and Scenarios. OOD: Begins by identifying objects and classes.</p>
12.	<p>i)Describe the golden rules for interface Design. (Nov 16)</p> <p>Theo Mandel [MAN97] coins three “golden rules”:</p> <ol style="list-style-type: none"> 1. Place the user in control. 2. Reduce the user’s memory load. 3. Make the interface consistent. <p>Place the User in Control</p> <ul style="list-style-type: none"> • Define interaction modes in a way that does not force a user into unnecessary or undesired actions. • Provide for flexible interaction • Allow user interaction to be interruptible and undoable. • Streamline interaction as skill levels advance and allow the interaction to be customized • Hide technical internals from the casual user • Design for direct interaction with objects that appear on the screen. <p>Reduce the User’s Memory Load</p> <ul style="list-style-type: none"> • Reduce demand on short-term memory. • Establish meaningful defaults. • Define shortcuts that are intuitive • The visual layout of the interface should be based on a real world metaphor. • Disclose information in a progressive fashion <p>Make the Interface Consistent</p> <ul style="list-style-type: none"> • Allow the user to put the current task into a meaningful context • Maintain consistency across a family of applications. • If past interactive models have created user expectations, do not make changes unless there is a compelling reason to do so. <p>ii)Explain Component level design with suitable examples. (Nov/Dec 16) (Nov/Dec 18)</p> <p>Component-level design, also called procedural design, occurs after data, architectural, and interface designs have been established. The intent is to translate the design model into operational software. But the level of abstraction of the existing design model is relatively high, and the abstraction level of the operational program is low. The translation can be challenging, opening the door to the introduction of subtle errors that are difficult to find and correct in later</p>

stages of the software process.

Graphical Design Notation

A flowchart is quite simple pictorially. A box is used to indicate a processing step.

A diamond represents a logical condition, and arrows show the flow of control. Figure 16.1 illustrates three structured constructs. The sequence is represented as two processing boxes connected by a line (arrow) of control. Condition, also called if-then-else, is depicted as a decision diamond that if true, causes then-part processing to occur, and if false, invokes else-part processing. Repetition is presented using two slightly different forms. The do-while tests a condition and executes a loop task repetitively as long as the condition holds true. A repeat-until executes the loop task first, then tests a condition and repeats the task until the condition fails. The selection (or select-case) construct shown in the figure is actually an extension of the if-then-else. A parameter is tested by successive decisions until a true condition occurs and a case-part processing path is executed.

Rules									
Conditions	1	2	3	4					n
Condition #1	✓			✓	✓				
Condition #2		✓		✓					
Condition #3			✓		✓				
Actions									
Action #1	✓			✓	✓				
Action #2		✓		✓					
Action #3			✓						
Action #4			✓	✓	✓				
Action #5	✓	✓			✓				

Decision tables provide a notation that translates actions and conditions (described in a processing narrative) into a tabular form.

Table is divided into four sections. The upper left-hand quadrant contains a list of all conditions. The lower left-hand quadrant contains a list of all actions that are possible based on combinations of conditions. The right-hand quadrants form a matrix that indicates condition combinations and the corresponding actions that will occur for a specific combination. Therefore, each column of the matrix may be interpreted as a processing rule.

The following steps are applied to develop a decision table:

1. List all actions that can be associated with a specific procedure (or module).
2. List all conditions (or decisions made) during execution of the procedure.
3. Associate specific sets of conditions with specific actions, eliminating impossible combinations of conditions; alternatively, develop every possible permutation of conditions.
4. Define rules by indicating what action(s) occurs for a set of conditions.

Program Design Language

Program design language (PDL), also called structured English or pseudocode, is "a pidgin language in that it uses the vocabulary of one language (i.e., English) and the overall syntax of another (i.e., a structured programming language)" [CAI75]. In this chapter, PDL is used as a generic reference for a design language.

At first glance PDL looks like a modern programming language. The difference between PDL and a real programming language lies in the use of narrative text (e.g., English) embedded directly within PDL statements.

Rules

Conditions	1	2	3	4	5
Fixed rate acct.	T	T	F	F	F
Variable rate acct.	F	F	T	T	F
Consumption <100 kwh	T	F	T	F	
Consumption ≥100 kwh	F	T	F	T	
Actions					
Min. monthly charge	✓				
Schedule A billing		✓	✓		
Schedule B billing				✓	
Other treatment					✓

A design language should have the following characteristics:

- A fixed syntax of keywords that provide for all structured constructs, data declaration, and modularity characteristics.
- A free syntax of natural language that describes processing features.
- Data declaration facilities that should include both simple (scalar, array) and complex (linked list or tree) data structures.
- Subprogram definition and calling techniques that support various modes of interface description.

13. Discuss about the design concepts in a software development process. (Nov 17)

A set of fundamental software design concepts has evolved over the history of software engineering. M. A. Jackson once said: "The beginning of wisdom for a [software engineer] is to recognize the difference between getting a program to work, and getting it right."

Abstraction

When we consider a modular solution to any problem, many levels of abstraction can be posed. At the highest level of abstraction, a solution is stated in broad terms using the language of the problem environment. At the lower levels of abstraction, a more detailed description of the solution is provided.

A procedural abstraction refers to a sequence of instructions that have a specific and limited function. And on this abstraction details are suppressed. An example of a procedural abstraction would be the word open for a door. Open implies a long sequence of procedural steps (e.g., walk to the door, reach out and grasp knob, turn knob and pull door, step away from moving door, etc.).

A data abstraction is a named collection of data that describes a data object. In the context of the procedural abstraction open, we define a data abstraction called **door**. Like any data object,

the data abstraction for **door** would encompass the set of attributes that describe the door (e.g., door type, swing direction, opening mechanism, weight, dimensions). It follows that the procedural abstraction open would make use of information contained in the attributes of the data abstraction **door**.

Architecture

Software architecture alludes to “the overall structure of the software and the ways in which that structure provides conceptual integrity for a system”. Architecture is the structure or organization of program components (modules), the manner in which these components interact, and the structure of data that are used by the components. 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 programs of program components. Framework models increase the level of design abstraction by attempting to identify repeatable architectural design frameworks that are encountered in similar types of application. Dynamic models address the behaviour 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 the business or technical process that the system must accommodate. Finally, functional models can be used to represent the functional hierarchy of a system.

Patterns

“A pattern is a named nugget of insight which conveys the essence of a proven solution to a recurring problem within a certain context amidst competing 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 functionality or structurally different pattern.

Modularity

Software architecture and design patterns embody modularity. “Modularity is a single attribute of software that allows a program to be intellectually manageable”. Monolithic software cannot be easily grasped by a software engineer. Consider two problems, p1 and p2. If the perceived complexity of p1 is greater than the perceived complexity of p2, it follows that the effort required to solve p1 is greater than the effort required to solve p2. As a general case, this result is intuitively obvious. It takes more time to solve a difficult problem. It also follows that the perceived complexity of two problems when they are combined is often greater than the sum of the perceived complexity when each is taken separately. This leads to “divide and conquer” strategy – it’s easier to solve a complex problem when you break it into manageable pieces. This has important implications with regard to modularity and software. If we subdivide software indefinitely, the effort required to develop it will become negligibly small.

Information hiding

The principle of information hiding suggests that modules be “characterized by design decisions that hide from all others.” Modules should be specified and designed so that information contained within a module is inaccessible to other modules that have no need for such information.

Functional Independence

The concept of functional independence is a direct outgrowth of modularity and the concepts of abstraction and information hiding. It is achieved by developing modules with “single minded” function and an “aversion” to exercise interaction with other modules. Independence

	<p>is assessed using two qualitative criteria: cohesion and coupling. Cohesion is an indication of the relative functional strength of a module. Coupling is a indication of the relative interdependence among modules.</p> <p>Refinement</p> <p>Stepwise refinement is a top-down design strategy. A program is developed successively refining levels of procedure detail. A hierarchy is developed by decomposing a macroscopic statement of function in a stepwise fashion until programming language statements are reached. Refinement is actually a process of elaboration. Abstraction and refinement are complementary concepts.</p> <p>Refactoring</p> <p>It is the process of a changing a software system in such a way that it does not alter the external behaviour of the code yet proves its internal structure. When software is refactored, then the existing design is examined for redundancy, unused design elements, inefficient or unnecessary algorithms, poorly constructed or inappropriate data structures, or any other design failure that can be corrected to yield a better design.</p> <p>Design Classes</p> <p>As the design model evolves, the software team must define a set of design classes that</p> <ul style="list-style-type: none"> • refine the analysis classes by providing design detail that will enable the classes to be implemented, and • create a new set of design classes that implement a software infrastructure to support the business solution. <p>Five different types of design classes are created by the designer. They are</p> <ul style="list-style-type: none"> • User interface classes • Business domain classes • Process classes • Persistent classes • System classes <p>Design classes can be reviewed to ensure that it is “well-formed”. So four characteristics of a well-formed design classes are</p> <ul style="list-style-type: none"> • Complete and sufficient • Primitiveness • High cohesion • Low Coupling
14.	<p>What is software architecture? Describe the different software architectural styles with examples (<i>Apr 19, 21</i>)</p> <p>Software architecture are the overall structure of the software and the ways in which that structure provides conceptual integrity for a system. In its simplest form, architecture is the hierarchical structure of program components (modules), the manner in which these components interact and the structure of data that are used by the components. One goal of software design is to derive an architectural rendering of a system. This rendering serves as a framework from which more detailed design activities are conducted. A set of architectural patterns enable a software engineer to reuse design level concepts.</p> <p>a set of properties that should be specified as part of an architectural design:</p> <p>Structural properties. This aspect of the architectural design representation defines the components of a system (e.g., modules, objects, filters) and the manner in which those components are packaged and interact with one another. For example, objects are packaged to encapsulate both data and the processing that manipulates the data and interact via the</p>

	<p>invocation of methods.</p> <p>Extra-functional properties. The architectural design description should address how the design architecture achieves requirements for performance, capacity, reliability, security, adaptability, and other system characteristics.</p> <p>Families of related systems. The architectural design should draw upon repeatable patterns that are commonly encountered in the design of families of similar systems. In essence, the design should have the ability to reuse architectural building blocks.</p>
15.	<p>Consider the problem of determining the number of different words in an input file. Carry out structured design by performing transform and transaction analysis construct the structured chart. (Apr 18)</p> <p style="text-align: center;">Algorithm Design Example - Counting Different Words</p> <pre> graph LR Input --> GetWordList((Get Word List)) GetWordList --> SortTheList((Sort The List)) SortTheList --> CountWords((Count the Number of Different Words)) CountWords --> PrintCount((Print the Count)) </pre> <p>• 1st PDL refinement of this Data Flow Graph:</p> <pre> count (in: file) returns integer var wl: word_list; begin sort (wl); count := different_words (wl); print (count); end; </pre>
16.	<p>What is the purpose of DFD? What are the components of DFD?</p> <p>An on-line shopping system for XYZ provides many services and benefits to its members and staffs. Currently, XYZ staffs manually handle the purchasing information with the use of basic office software, such as Microsoft Office Word and Excel. It may results in having mistakes easily and the process is very inconvenient. XYZ needs an online shopping system at their Intranet based on the requirements of users. XYZ online shopping system has five key features:</p> <ol style="list-style-type: none"> To provide the user friendly online shopping cart function to members to replace hard copy ordering form; To store inventory and sales information in database to reduce the human mistakes, increase accuracy and enhance the flexibility of information processing; To provide an efficient inventory system which can help the XYZ staffs to gain enough information to update the inventory; To be able to print invoices to members and print a set of summary reports for XYZ's internal usage. To design the system that is easy to maintain and upgrade.
<p style="text-align: center;">UNIT IV TESTING AND MAINTENANCE</p> <p>Software testing fundamentals-Internal and external views of Testing-white box testing - basis path testing-control structure testing-black box testing- Regression Testing - Unit Testing - Integration Testing - Validation Testing - System Testing And Debugging -Software Implementation Techniques: Coding practices-Refactoring-Maintenance and Reengineering-BPR model-Reengineering process model-Reverse and Forward Engineering.</p>	

UNIT IV/ PART- A	
1	<p>Distinguish between stress and load testing. (Apr 12)</p> <p>Stress testing executes a system in a manner that demands resources in abnormal quality, frequency or volume. Load testing is a testing in which real world loading is tested at a variety of load levels and in a variety of combinations.</p>
2	<p>What is boundary condition testing? (Apr 12)</p> <p>A greater number of errors occurs at the boundaries of the input domain rather than in "center". It is a test case design that complements equivalence partitioning.</p>
3	<p>Define black box testing. (Apr 12)</p> <p>Black-box testing is also called behavioral testing which focuses on the functional requirements of the software. This testing enables the software engineer to derive sets of input conditions that will fully exercise all functional requirements for a program.</p>
4	<p>What is the purpose of regression testing?(Nov 11)</p> <p>Regression testing is the re-execution of some subset of tests that have already been conducted to ensure that changes have not propagated unintended side effects. The regression testing suite contains the following different test cases</p> <ul style="list-style-type: none"> • A representative sample of tests that will exercise all software functions • Additional tests that focus on software functions that are likely to be affected by the change. • Tests that focus on the software components that have been changed.
5	<p>What is integration testing? (Apr 11)</p> <p>Integration testing is a systematic technique for constructing the software architecture while at the same time conducting tests to uncover errors associated with interfacing. The objective is to take unit tested components and build a program structure that has been dictated by design.</p>
6	<p>What is regression testing? (Apr 11, 15 , Nov 13, 18)</p> <p>Regression testing is the re-execution of some subset of tests that have already been conducted to ensure that changes have not propagated unintended side effects.</p>
7	<p>Define error, fault and failure. (Nov 10)</p> <ul style="list-style-type: none"> • Error: A discrepancy between a computed, observed value and the true, specified correct value or condition. • Failure: The inability of a system or component to perform its requirements functions within specified requirements. • Fault: An incorrect step, process or definition in a computer program which causes the program in an unintended or unanticipated manner.
8	<p>What are the levels at which testing is done? (Nov 13) or List the level of testing (Apr 19)</p> <ul style="list-style-type: none"> • Functional testing • Non-Functional testing
9	<p>What are the various testing activities?</p> <ul style="list-style-type: none"> • Test planning • Test case design • Test execution • Data collection and Effective evaluation
10	<p>Give the two major groupings of software testing?</p> <ul style="list-style-type: none"> • Black Box Testing: This approach focuses on inputs, outputs, and principle function of

	<p>software module.</p> <ul style="list-style-type: none"> • Glass Box Testing: This approach looks into the structure of code for a software module.
11	<p>What is data flow testing?</p> <p>The data flow testing method selects test paths of a program according to the locations of definitions and uses of variables in the program.</p>
12	<p>Give the major types of software testing?</p> <p>Functional tests, Performance tests, Stress tests, Testing in the small , Testing in the large, Black box testing and White (glass) box testing.</p>
13	<p>What are the objectives of testing?</p> <ul style="list-style-type: none"> • Testing is a process of executing a program with the intend of finding an error. • A good test case is one that has high probability of finding an undiscovered error. • A successful test is one that uncovers as an-yet undiscovered error.
14	<p>What are the characteristics of a good tester? (Nov 10)</p> <p>The characteristics of a good tester are</p> <ul style="list-style-type: none"> • A software team should conduct effective formal technical reviews. Many errors may be eliminated before testing commences. • Testing should begin at 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 an independent test group. • Testing and debugging must be accommodated in any test strategy.
15	<p>What is the purpose of stress tests?</p> <p>Stress tests are designed to overload a system in ways. Examples of stress tests include attempting to sign or more than the maximum number of allowed terminals, processing more than the allowed number of identifies or static levels or disconnecting a communication links.</p>
16	<p>What are the contents of maintenance guide?</p> <p>A maintenance guide provides a technical description of the operational capabilities of the entire system and hierarchy diagrams, call graphs and cross reference directories for the system.</p>
17	<p>Give some examples of functional boundary tests.</p> <p>Examples of functional boundary tests include testing real-valued square root routines with small positive numbers, zero and negative numbers or testing matrix inversion routine on a one-by-one matrix and a singular matrix.</p>
18	<p>What are the errors commonly found during Unit testing?</p> <p>Misunderstood or incorrect arithmetic precedence, Mixed mode operation. Incorrect initialization, Precision inaccuracy, Incorrect symbolic representation of an expression</p>
19	<p>What are Validation and Conditional testing? (Nov 14)</p> <p>Validation testing is the process to demonstrate that the product fulfills its intended use when deployed on appropriate environment.</p> <p>Conditional testing is a test case design method that exercise the logical conditions contained in a program module.</p>
20	<p>What are the main steps followed in the testing scheme?</p> <p>Select what is to be measured by the test, Decide how whatever is being is to be tested, develop the test cases, Determine what the expected results of the test, Execute the test cases, Compare</p>

	the results.		
21	<p>What is system testing?</p> <p>Software and hardware are integrated and a full range of system tests are conducted in an attempt to uncover errors at the software and hardware interfaces. Most real system process interrupts. Therefore testing the handling of these Boolean events is essential.</p>		
22	<p>What is the overall strategy for the software testing?</p> <p>A Strategy for software testing may also be viewed in the context of the spiral. Unit test begins at the vertex of the spiral and concentrates on each unit of the software as implemented in the source code. Testing progresses by moving outward along the spiral to integration testing, where the focus is on design and the construction of the software architecture.</p>		
23	<p>What is security testing?</p> <p>Security testing attempts to verify that protection mechanisms built into a system will in fact protect it from improper penetration. The system security is to of course be tested for invulnerability from frontal attack but must also be tested for invulnerability from flank or rear attack.</p>		
24	<p>What is Code Refactoring? What are best practices for “CODING”?(Apr 15)</p> <p>Code refactoring is the process of restructuring existing computer code without changing its external behaviour.</p> <p>Refactoring improves non-functional attributes of the software.</p> <p>The best practices for CODING are:</p> <ul style="list-style-type: none"> • To know what the code block must perform • Indicate a brief description of what a variable is for (reference to commenting) • To correct errors as they occur. • Keep your code simple • Maintain naming conventions which are uniform throughout. 		
25	<p>How can refactoring be made more effective? (Apr 16)</p> <ul style="list-style-type: none"> • When refactoring, we maintain the behavior of the current system. • When refactoring, we begin with the existing code and change it over time as needed. • When refactoring, we can steadily improve existing code over time without disrupting the release schedule. • When refactoring, we can start with less up-front work, change direction when needed and stop part-way through if that’s appropriate. • When refactoring, we often support parts of the old and new system at the same time. 		
26	<p>Why does software fail after it has passed from acceptance testing? (Apr 16)</p> <p>Acceptance criteria may not require the software to be 100% bug-free. Test cases may not be able to test for everything that could possibly go wrong. (It is difficult to make things idiot proof because idiots are SO creative!)Interactions with new (or unexpected) hardware and third party software might cause the software in question to fail. The acceptance tests were not rigorous enough.</p>		
26	<p>What is the difference between verification and validation? Which types of testing address verification? Which type of testing address validation?(Nov 16, 17 , Apr 18)</p> <table border="1"> <tr> <td>Validation</td><td>Verification</td></tr> </table>	Validation	Verification
Validation	Verification		

	The process of checking that a system meets the needs and expectations of the customer.	The process of checking that a system meets its specification
	Checks "Are we building the right product"?	Checks "Are we building the product right"?
	Includes all the dynamic testing techniques.	Involves all the static testing techniques
	Involves activities like Unit, Integration and System testing.	Involves activities like Reviews, Walkthroughs and Inspections.
27	What is "Smoke Testing"? (Apr 17) Smoke testing is an integration testing approach that is commonly used when "shrink wrapped" software products are being developed. It is designed as a pacing mechanism for time critical projects, allowing the software team to assess its project on a frequent basis.	
28	List two testing strategies that address verification .which types of testing address validation and verification? (Apr 17) Verification-Includes all the dynamic testing techniques. Involves Unit, Integration and System testing are addresses verification. Validation- Involves all the static testing techniques. Alpha testing, Beta testing and acceptance testing addresses verification.	
29	Mention the purpose of stub and driver use for testing. (Nov 17) Stubs are dummy modules that are always distinguish as "called programs", or you can say that is handle in integration testing (top down approach), it used when sub programs are under construction. Stubs are considered as the dummy modules that always simulate the low level modules. Drivers are also considered as the form of dummy modules which are always distinguished as "calling programs", that is handled in bottom up integration testing, it is only used when main programs are under construction. Drivers are considered as the dummy modules that always simulate the high level modules.	
30	What are the testing principles the software engineer must apply while performing the software testing? (Apr 18) <ul style="list-style-type: none"> • All tests should be trace able to customer requirements. • Tests should be planned long before testing begins. • The pareto principle can be applied to software testing-80% of all errors uncovered during testing will likely be traceable to 20% of all program modules. • Testing should begin "in the small" and progress toward testing "in the large". • Exhaustive testing is not possible. To be most effective, an independent third party should conduct testing.	
31	Define restructuring. Software restructuring modifies source code and/or data in an effort to make it amenable to future changes. In general, restructuring does not modify the overall program architecture. It tends to focus on the design details of individual modules and on local data structures defined within modules.	
32	What is Forward Engineering? Forward engineering would be rebuilt using a automated "reengineering engine." The old program would be fed into the engine, analyzed, restructured, and then regenerated in a form that exhibited the best aspects of software quality.	
33	What is Reverse Engineering? (Apr 19)	

	Reverse engineering for software is the process of analyzing a program in an effort to create a representation of the program at a higher level of abstraction than source code. Reverse engineering is a process of design recovery. Reverse engineering tools extract data, architectural, and procedural design information from an existing program. successful reverse engineering derives one or more design and manufacturing specifications for a product by examining actual specimens of the product.
35	Define verification, validation testing and debugging. (Nov 17) <ul style="list-style-type: none"> • Verification is the process, to ensure that whether we are building the product right. • Validation is the process, whether we are building right product. • Debugging is the process of finding and fixing software coding errors
36	Mention the software testability checklist (Apr 21) <ul style="list-style-type: none"> • Create System and Acceptance Tests. • Start Acceptance test Creation. • Identify test team. • Create Workplan. • Create test Approach. • Link Acceptance Criteria and Requirements to form the basis of acceptance test. • Use subset of system test cases to form requirements portion of acceptance test.
37	How black box testing is differing from white box testing ? (Apr 21) <p>Black Box Testing is a software testing method in which the internal structure/ design/ implementation of the item being tested is not known to the tester. White Box Testing is a software testing method in which the internal structure/ design/ implementation of the item being tested is known to the tester.</p>
UNIT IV/ PART- B	
1.	<p>Given a set of numbers 'n', the function Find Prime(a[],n) prints a number- if it is a prime number. Draw a control flow graph, calculate the cyclomatic complexity and enumerate all paths. State how many test case-s are- needed to adequately cover the code in terms of branches, decisions and statement? Develop the necessary test cases using sample values for 'a' and 'n'. (Nov 13)</p> <p>Ans:-Software metrics can be defined as the continuous application of measurement based techniques to the software development process and its products to supply meaningful and timely management information together with the use of those techniques to improve that process and its products.</p> <p>cyclomatic complexity is a measure of the complexity of a module's decision structure. It is the number of linearly independent paths and the minimum number of paths to be tested. Actual complexity is the number of independent paths traversed during testing.</p> <p>Module design complexity metric is the complexity of design-reduced module and reflects the complexity of the module's calling patterns to its subordinate modules. This metric differentiates between modules and complicates the design of the program.</p> <p>A control flow graph (CFG) in computer science is a <u>representation</u>, using <u>graph</u> notation, of all paths that might be traversed through a <u>program</u> during its <u>execution</u>.</p> <p>In a control flow graph each node in the graph represents a basic block, i.e. a straight-line piece of code without any jumps or jump targets; jump targets start a block, and jumps end a block. Directed edges are used to represent jumps in the control flow. There are, in most</p>

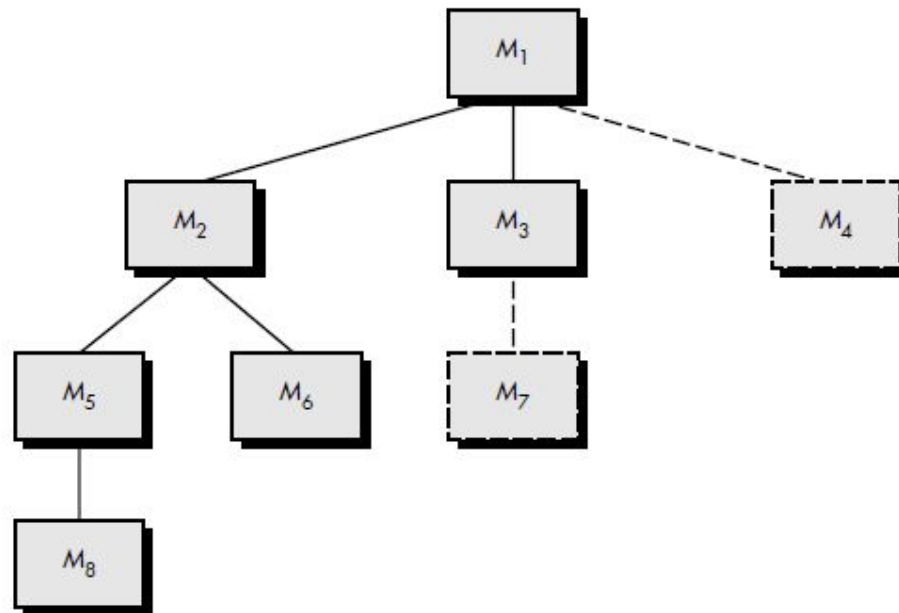
	<p>presentations, two specially designated blocks: the entry block, through which control enters into the flow graph, and the exit block, through which all control flow leaves.</p> <p>Because of its construction procedure, in a non-trivial CFG (i.e. one with more than zero edges), every edge $A \rightarrow B$ has the property that:</p> <p>Out degree(A) > 1 or in degree(B) > 1 (or both).</p> <p>Consider the following fragment of code:</p> <div><pre>0: (A) x = read_n For (i=2,i<n/2,i++) 1: (A) if (n%i)== 0 2: (B) print n+ " is prime." 3: (B) go to 5 4: (C) print n + " is composite." 5: (D) end program</pre></div> <p>In the above, we have 4 basic blocks: A from 0 to 1, B from 2 to 3, C at 4 and D at 5. In particular, in this case, A is the "entry block", D the "exit block" and lines 4 and 5 are jump targets. A graph for this fragment has edges from A to B, A to C, B to D and C to D.</p> <p>Based on this a control flow graph with logic involved .</p> <p>Pre-condition: let the prime no be n case1: expected o/p (prime no)result divide the no n by 1 remainder=0 pass divide the no n by n remainder=0 pass</p> <p>divide the no n by 2 remainder!=0 pass . . divide the no n by up to n-1 and if remanider not equal zero then it is a prime no. Approximately 4 test cases can be roughly designed and tested.</p> <table><thead><tr><th>Test case ID</th><th>test condition</th><th>Result</th></tr></thead><tbody><tr><td>1</td><td>1</td><td>true</td></tr><tr><td>2</td><td>3</td><td>true</td></tr><tr><td>3</td><td>4</td><td>false</td></tr><tr><td>4</td><td>5</td><td>true</td></tr></tbody></table> <p>Like this many other test cases can be formulated and output can be determined.</p>	Test case ID	test condition	Result	1	1	true	2	3	true	3	4	false	4	5	true
Test case ID	test condition	Result														
1	1	true														
2	3	true														
3	4	false														
4	5	true														
2.	<p>Illustrate black-box testing with an example. Write short notes on unit testing and debugging. (Apr 10, 15)</p> <p>Ans:- Black box testing is a software testing technique that focuses on the analysis of software functionality ,versus internal system mechanism.It was developed as a method of analyzing client requirements ,specifications and high level strategies.</p> <p>A black box software tester selects a set of valid and invalid input and code execution conditions and checks for valid output responses. A black box software tester selects a set of valid and invalid input and code execution conditions and checks for valid output</p>															

	<p>responses.</p> <p>Black box testing is also known as functional testing.</p> <p>(ii) What is Equivalence Class Partitioning? List rules used to define valid and invalid equivalence classes. Explain the technique using examples.</p> <p>Ans:- Equivalence class partitioning :-If a tester is viewing the software under test as a black box with well-defined inputs and outputs, a good approach to selecting test inputs is to use a method called equivalence partitioning.</p> <p>It results in a partitioning of the input domain of the software under test. The technique can also be used to partition the output domain.</p> <p>The finite number of partitions on equivalence classes that result allow the tester to select a member of the equivalence class as a representative of the class.</p> <p>How do testers identify equivalence classes?:-</p> <p>One approach is to use a set of what Glen Myers calls “interesting input conditions”. The input conditions usually come from a description in the specification of the software to be tested. The tester uses the conditions to partition the input domain into equivalence classes and then develops test cases to cover all the classes.</p> <p>(iii)What is Boundary Value Analysis? Explain the technique specifying rules and its usage with the help of an example. (NOV/DEC2013)</p> <p>Ans:- Boundary Value Analysis:-</p> <p>A greater number of errors occurs at the boundaries of the input domain rather than in the “center.” It is for this reason that boundary value analysis (BVA) has been developed as a testing technique. Boundary value analysis leads to a selection of test cases that exercise bounding values.</p> <p>Boundary value analysis is a test-case design technique that complements equivalence partitioning. Rather than selecting any element of an equivalence class, BVA leads to the selection of test cases at the “edges” of the class. Rather than focusing solely on input conditions, BVA derives test cases from the output domain as well</p> <p>Guidelines for BVA are similar in many respects to those provided for equivalence partitioning:-</p> <ol style="list-style-type: none"> 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 1 and 2 to output conditions. For example, assume that a temperature versus pressure table 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. 4. If internal program data structures have prescribed boundaries (e.g., a table has a defined limit of 100 entries), be certain to design a test case to exercise the data structure at its boundary.
3.	<p>What is white box testing? Explain how basis path testing helps to derive test cases to test every statement of a program. (Nov 12 , Apr 15, 17)</p> <p>Ans:- Black-box testing, also called behavioral testing, focuses on the functional requirements of the software. That is, black-box testing techniques enable you to derive sets of input conditions that will fully exercise all functional requirements for a program. Black-box testing is not an alternative to white-box techniques. Rather, it is a complementary approach that is likely to uncover a different class of errors than white box testing.</p> <p>Black-box testing attempts to find errors in the following categories: (1) incorrect or missing</p>

	<p>functions, (2) interface errors, (3) errors in data structures or external database access, (4) behavior or performance errors, and (5) initialization and termination errors.</p> <p>Unlike white-box testing, which is performed early in the testing process, black box testing tends to be applied during later stages of testing .Because black-box testing purposely disregards control structure, attention is focused on the information domain. Tests are designed to answer the following questions:-</p> <ul style="list-style-type: none"> • How is functional validity tested? • How are system behavior and performance tested? • What classes of input will make good test cases? • Is the system particularly sensitive to certain input values? • How are the boundaries of a data class isolated? • What data rates and data volume can the system tolerate? • What effect will specific combinations of data have on system operation? <p>The concept of Black-box testing can be explained with model-based testing as an example:-.</p> <p>Model-based testing (MBT) is a black-box testing technique that uses information contained in the requirements model as the basis for the generation of test cases. In many cases, the model-based testing technique uses UML state diagrams, an element of the behavioral model, as the basis for the design of test cases.</p> <p>The MBT technique requires five steps:</p> <ul style="list-style-type: none"> ➤ Analyze an existing behavioral model for the software or create one. <p>Recall that a behavioral model indicates how software will respond to external events or stimuli. To create the model, you should perform the steps:-</p> <ul style="list-style-type: none"> • evaluate all use cases to fully understand the sequence of interaction within the system, • identify events that drive the interaction sequence and understand how these events relate to specific objects, • create a sequence for each use case, • build a UML state diagram for the system and • review the behavioral model to verify accuracy and consistency. <ul style="list-style-type: none"> ➤ Traverse the behavioral model and specify the inputs that will force the software to make the transition from state to state. The inputs will trigger events that will cause the transition to occur. ➤ Review the behavioral model and note the expected outputs as the software makes the transition from state to state. Recall that each state transition is triggered by an event and that as a consequence of the transition, some function is invoked and outputs are created. For each set of inputs (test cases) you specified in step 2, specify the expected outputs as they are characterized in the behavioral model. “A fundamental assumption of this testing is that there is some mechanism, a test oracle, that will determine whether or not the results of a test execution are correct”. In essence, a test oracle establishes the basis for any determination of the correctness of the output. In most cases, the oracle is the requirements model, but it could also be another document or application, data recorded else where, or even a human expert. ➤ Execute the test cases. Tests can be executed manually or a test script can be created and executed using a testing tool. ➤ Compare actual and expected results and take corrective action as required. MBT helps to uncover errors in software behavior, and as a consequence, it is extremely useful when testing event-driven applications. ➤ Model-based testing can also be used when software.
4.	<p>(i) Define regression testing.</p> <p>(ii) Distinguish top down and bottom-up integration.</p>

	<p>(iii) How testing different from debugging? Justify. (Nov 10)</p> <p>Ans:-White box testing is a complimentary approach to test design where the tester has knowledge of the internal logic structure of the software under test.</p> <p>Goal:-The tester's goal is to determine if all the logic and data elements in the software unit function properly. this is called white box or glass box approach to test case design. The knowledge needed for white box testing design approach becomes available to the tester in the later phases of the software life cycle specifically during the detailed design phase of development. It follows black box testing as the test efforts for the given project progress in time.</p> <p>Goals of white box testing:-</p> <ul style="list-style-type: none"> • To ensure that the internal components of a program are functioning properly. • A common focus is on structural elements like statements and branches. • He tester develops test cases that exercise these structural elements to determine if defects exist in the program structure. • By exercising all of the selected structural elements, the tester hopes to improve the chances of detecting defects. <p>Test Adequacy Criteria:-</p> <p>Testers need a framework to decide which structural elements to select as a focus of testing, for choosing appropriate test data and for deciding when the testing efforts are adequate enough to terminate the process with confidence that the software is working properly. such a framework exists in the form of test adequacy criteria.</p> <p>If a test Adequacy criteria focuses on the structural properties of a program, it is called program based adequacy criterion.</p> <p>They use either:-</p> <ul style="list-style-type: none"> • logical & control structures (or) • program text (or) • data flow (or) • faults as the focal point of an adequacy evaluation <p>Coverage Analysis:-</p> <p>The concept of test data adequacy criteria and the requirement that certain properties or features of the code are to be exercised by test cases, leads to an approach called coverage analysis which in practice I used:- to set testing signals, to develop & evaluate test data.</p> <ul style="list-style-type: none"> • The logic elements mostly used in coverage are based on the flow of control in a unit of code:- They are :-program statements, decision/branches, conditions, combinations of decisions & conditions and paths.
5.	<p>Explain in detail about integration testing. (Apr 13,14) (OR) Describe about the various Integration and Debugging strategies followed in Software development. What is integration testing? Discuss any one method in detail. (Apr 15, 18, Nov 14, 18) (OR) Explain top down integration testing with an example.</p> <p>A neophyte in the software world might ask a seemingly legitimate question once all modules have been unit tested: "If they all work individually, why do you doubt that they'll work when we put them together?" The problem, of course, is "putting them together" – interfacing. Data can be lost across an interface; one component can have an inadvertent, adverse effect on another; sub functions, when combined, may not produce the desired major function; individually acceptable imprecision may be magnified to unacceptable levels; global data structures can present problems. Sadly, the list goes on and on.</p> <p>Integration testing is a systematic technique for constructing the software architecture while at the same time conducting tests to uncover errors associated with interfacing. The objective is to take unit-tested components and build a program structure that has been dictated by design.</p>

There is often a tendency to attempt non incremental integration; that is, to construct the program using a “big bang” approach. All components are combined in advance. The entire program is tested as a whole. And chaos usually results! A set of errors is encountered. Correction is difficult because isolation of causes is complicated by the vast expanse of the entire program. Once these errors are corrected, new ones appear and the process continues in a seemingly endless loop. Incremental integration is the antithesis of the big bang approach. The program is constructed and tested in small increments, where errors are easier to isolate and correct; interfaces are more likely to be tested completely; and a systematic test approach may be applied. In the paragraphs that follow, a number of different incremental integration strategies are discussed.



Top-down integration. Top-down integration testing is an incremental approach to construction of the software architecture. Modules are integrated by moving downward through the control hierarchy, beginning with the main control module. Modules subordinate (and ultimately subordinate) to the main control module are incorporated into the structure in either a depth-first or breadth-first manner.

Referring to the Figure depth-first integration integrates all components on a major control path of the program structure. Selection of a major path is somewhat arbitrary and depends on application-specific characteristics. For example, selecting the left-hand path, components M1, M2, M5 would be integrated first. Next, M8 or (if necessary for proper functioning of M2) M6 would be integrated. Then, the central and right-hand control paths are built.

Breadth-first integration incorporates all components directly subordinate at each level, moving across the structure horizontally. From the figure, components M2, M3, and M4 would be integrated first. The next control level, M5, M6, and so on, follow.

The integration process is performed in a series of five steps:

1. The main control module is used as a test driver and stubs are substituted for all components directly subordinate to the main control module.
2. Depending on the integration approach selected (i.e., depth or breadth first), subordinate stubs are replaced one at a time with actual components.
3. Tests are conducted as each component is integrated.
4. On completion of each set of tests, another stub is replaced with the real component.
5. Regression testing (discussed later in this section) may be conducted to ensure that new errors have not been introduced. The process continues from step 2 until the entire program

	structure is built.
6.	<p>Explain unit testing in detail. (<i>Nov 09, 18 , Apr 13</i>)</p> <p>Ans:- Unit testing focuses verification effort on the smallest unit of software design—the software component or module. Using the component-level design description as a The relative complexity of tests and the errors those tests uncover is limited by the constrained scope established for unit testing. The unit test focuses on the internal processing logic and data structures within the boundaries of a component. This type of testing can be conducted in parallel for multiple components.</p> <p>Unit-test considerations:-</p> <p>The module interface is tested to ensure that information properly flows into and out of the program unit under test. Local data structures are examined to ensure that data stored temporarily maintains its integrity during all steps in an algorithm's execution. All independent paths through the control structure are exercised to ensure that all statements in a module have been executed at least once. Boundary conditions are tested to ensure that the module operates properly at boundaries established to limit or restrict processing. And finally, all error-handling paths are tested. Data flow across a component interface is tested before any other testing is initiated. If data do not enter and exit properly, all other tests are moot. In addition, local data structures should be exercised and the local impact on global data should be ascertained (if possible) during unit testing.</p> <p>Selective testing of execution paths is an essential task during the unit test. Test cases should be designed to uncover errors due to erroneous computations, incorrect comparisons, or improper control flow.</p> <p>Boundary testing is one of the most important unit testing tasks. Software often fails at its boundaries. That is, errors often occur when the <i>n</i>th element of an <i>n</i>-dimensional array is processed, when the <i>i</i>th repetition of a loop with <i>i</i> passes is invoked, when the maximum or minimum allowable value is encountered. Test cases that exercise data structure, control flow, and data values just below, at, and just above maxima and minima are very likely to uncover errors.</p> <p>A good design anticipates error conditions and establishes error-handling paths to reroute or cleanly terminate processing when an error does occur. Yourdon [You75] calls this approach anti bugging. Unfortunately, there is a tendency to incorporate error handling into software and then never test it. A true story may serve to illustrate:</p> <p>A computer-aided design system was developed under contract. In one transaction processing module, a practical joker placed the following error handling message after a series of conditional tests that invoked various control flow branches: ERROR! THERE IS NO WAY YOU CAN GET HERE. This "error message" was uncovered by a customer during user training!</p> <p>Among the potential errors that should be tested when error handling is evaluated are: (1) error description is unintelligible, (2) error noted does not correspond to error encountered, (3) error condition causes system intervention prior to error handling, (4) exception-condition processing is incorrect, or (5) error description does not provide enough information to assist in the location of the cause of the error.</p> <p>Unit-test procedures.-</p> <p>Unit testing is normally considered as an adjunct to the coding step. The design of unit tests can occur before coding begins or after source code has been generated. A review of design information provides guidance for establishing test cases that are likely to uncover errors in each of the categories discussed earlier. Each test case should be coupled with a set of expected results.</p>

	<p>Because a component is not a stand-alone program, driver and/or stub software must often be developed for each unit test. The unit test environment is illustrated in Figure 17.4. In most applications a driver is nothing more than a “main program” that accepts test case data, passes such data to the component (to be tested), and prints relevant results. Stubs serve to replace modules that are subordinate (invoked by) the component to be tested. A stub or “dummy subprogram” uses the subordinate module’s interface, may do minimal data manipulation, prints verification of entry, and returns control to the module undergoing testing.</p> <p>Drivers and stubs represent testing “overhead.” That is, both are software that must be written (formal design is not commonly applied) but that is not delivered with the final software product. If drivers and stubs are kept simple, actual overhead is relatively low. Unfortunately, many components cannot be adequately unit tested with “simple” overhead software. In such cases, complete testing can be postponed until the integration test step (where drivers or stubs are also used). Unit testing is simplified when a component with high cohesion is designed. When only one function is addressed by a component, the number of test cases is reduced and errors can be more easily predicted and uncovered.</p>
7.	<p>Explain in detail about Basis path testing and validation testing? (<i>Nov 10 , Apr 14</i>)</p> <p>Validation testing begins at the culmination of integration testing, when individual components have been exercised, the software is completely assembled as a package, and interfacing errors have been uncovered and corrected. At the validation or system level, the distinction between conventional software, object-oriented Validation can be defined in many ways, but a simple (albeit harsh) definition is that validation succeeds when software functions in a manner that can be reasonably expected by the customer. At this point a battle-hardened software developer might protest: “Who or what is the arbiter of reasonable expectations?” If a Software Requirements Specification has been developed, it describes all user-visible attributes of the software and contains a Validation Criteria section that forms the basis for a validation-testing approach.</p> <p>Validation-Test Criteria:</p> <p>Software validation is achieved through a series of tests that demonstrate conformity with requirements. A test plan outlines the classes of tests to be conducted, and a test procedure defines specific test cases that are designed to ensure that all functional requirements are satisfied, all behavioral characteristics are achieved, all content is accurate and properly presented, all performance requirements are attained, documentation is correct, and usability and other requirements are met (e.g., transportability, compatibility, error recovery, maintainability). After each validation test case has been conducted, one of two possible conditions exists: (1) The function or performance characteristic conforms to specification and is accepted or (2) a deviation from specification is uncovered and a deficiency list is created. Deviations or errors discovered at this stage in a project can rarely be corrected prior to scheduled delivery. It is often necessary to negotiate with the customer.</p> <p>Basis path testing is a white-box testing technique first proposed by Tom McCabe. The basis path method enables the test-case designer to derive a logical complexity measure of a procedural design and use this measure as a guide for defining a basis set of execution paths. Test cases derived to exercise the basis set are guaranteed to execute every statement in the program at least one time during testing.</p> <p>Flow Graph Notation:</p> <p>Before we consider the basis path method, a simple notation for the representation of control flow, called a flow graph (or program graph) must be introduced. Each structured construct has a corresponding flow graph symbol.</p> <p>3 In actuality, the basis path method can be conducted without the use of flow graphs. However, they serve as a useful notation for understanding control flow and illustrating the</p>

	<p>approach. Here, a flowchart is used to depict program control structure. maps the flowchart into a corresponding flow graph (assuming that no compound conditions are contained in the decision diamonds of the flowchart). Each circle, called a flow graph node, represents one or more procedural statements. A sequence of process boxes and a decision diamond can map into a single node. The arrows on the flow graph, called edges or links, represent flow of control and are analogous to flowchart arrows. An edge must terminate at a node, even if the node does not represent any procedural statements (e.g., see the flow graph symbol for the if-then-else construct). Areas bounded by edges and nodes are called regions. When counting regions, we include the area outside the graph as a region.</p>
8.	<p>(i)What is black box testing?</p> <p>(ii)What is Equivalence Class Partitioning? List rules used to define valid and invalid equivalence classes. Explain the technique using examples.</p> <p>(iii)What is Boundary Value Analysis? Explain the technique specifying rules and its usage with the help of an example. (Nov 13, 16)</p> <p>Ans:- Black box testing is a software testing technique that focuses on the analysis of software functionality ,versus internal system mechanism.It was developed as a method of analyzing client requirements ,specifications and high level strategies.</p> <p>A black box software tester selects a set of valid and invalid input and code execution conditions and checks for valid output responses. A black box software tester selects a set of valid and invalid input and code execution conditions and checks for valid output responses.</p> <p>Black box testing is also known as functional testing.</p> <p>(ii) What is Equivalence Class Partitioning? List rules used to define valid and invalid equivalence classes. Explain the technique using examples.</p> <p>Ans:- Equivalence class portioning :-If a tester is viewing the software under test as a black box with well-defined inputs and outputs, a good approach to selecting test inputs is to use a method called equivalence partitioning.</p> <p>It results in a partitioning of the input domain of the software under test..The technique can also be used to partition the output domain.</p> <p>The finite number of partitions on equivalence classes that result allow the tester to select a member of the equivalence class as a representative of the class.</p> <p>How do testers identify equivalence classes?:-</p> <p>One approach is to use a set of what Glen Myers calls “interesting input conditions”. The input conditions usually come from a description in the specification of the software to be tested.The tester uses the conditions to partition the input domain into equivalence classes and then develops test cases to cover all the classes.</p> <p>(iii)What is Boundary Value Analysis? Explain the technique specifying rules and its usage with the help of an example. (NOV/DEC2013)</p> <p>Ans:- Boundary Value Analysis:-</p> <p>A greater number of errors occurs at the boundaries of the input domain rather than in the “center.” It is for this reason that boundary value analysis (BVA) has been developed as a testing technique. Boundary value analysis leads to a selection of test cases that exercise bounding values.</p> <p>Boundary value analysis is a test-case design technique that complements equivalence partitioning. Rather than selecting any element of an equivalence class, BVA leads to the selection of test cases at the “edges” of the class. Rather than focusing solely on input conditions, BVA derives test cases from the output domain as well</p> <p>Guidelines for BVA are similar in many respects to those provided for equivalence partitioning:-</p>

	<p>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.</p> <p>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.</p> <p>3. Apply guidelines 1 and 2 to output conditions. For example, assume that a temperature versus pressure table 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.</p> <p>4. If internal program data structures have prescribed boundaries (e.g., a table has a defined limit of 100 entries), be certain to design a test case to exercise the data structure at its boundary.</p>
9.	<p>What is the purpose of testing? Explain clearly several testing strategies for conventional softwares and object-oriented softwares. (<i>Nov 10</i>)</p> <p>Testing is the process of exercising a s/w component with a selected set of test cases with the intention of revealing defects, to evaluate quality.</p> <p>The following are the testing strategies:-</p> <ul style="list-style-type: none"> • When the test objective is to detect defects, then a good test-case is one that has high probability of revealing a yet-undetected defect. • Test result should be inspected meticulously. • A test case should contain the expected output. • Test cases should be developed for both valid and invalid input conditions. • The probability of detecting additional defects in a s/w component is directly proportional to the no of defects already detected in that component. • Testing should be carried out by a group that is independent of the development group. • It should be re-usable and repeatable. • Testing should be planned. • Testing activities should be integrated into the s/w development life cycle. • It is a creative and challenging task.
10.	<p>What is system testing? Discuss types of system tests. (<i>Nov 14</i>)</p> <p>System Testing (ST) is a black box testing technique performed to evaluate the complete system the system's compliance against specified requirements. In System testing, the functionalities of the system are tested from an end-to-end perspective.</p> <p>System testing is performed on the entire system in the context of a Functional Requirement Specification(s) (FRS) and/or a System Requirement Specification (SRS). System testing tests not only the design, but also the behaviour and even the believed expectations of the customer. It is also intended to test up to and beyond the bounds defined in the software/hardware requirements specification(s). Some of system testing are as follows:</p> <p>System testing is actually a series of different tests whose primary purpose is to fully exercise the computer-based system. Although each test has a different purpose, all work to verify that system elements have been properly integrated and perform allocated functions.</p> <p>Recovery testing is a system test that forces the software to fail in a variety of ways and verifies that recovery is properly performed. If recovery is automatic (performed by the system itself), re-initialization, checkpointing mechanisms, data recovery, and restart are evaluated for correctness. If recovery requires human intervention, the mean-time-to-repair (MTTR) is evaluated to determine whether it is within acceptable limits.</p> <p>Security Testing</p> <p>Security testing attempts to verify that protection mechanisms built into a system</p> <p>Stress testing executes a system in a manner that demands resources in abnormal quantity,</p>

	<p>frequency, or volume. For example, (1) special tests may be designed that generate ten interrupts per second, when one or two is the average rate, (2) input data rates may be increased by an order of magnitude to determine how input functions will respond, (3) test cases that require maximum memory or other resources are executed, (4) test cases that may cause thrashing in a virtual operating system are designed, (5) test cases that may cause excessive hunting for disk-resident data are created. Essentially, the tester attempts to break the program.</p> <p>Performance testing occurs throughout all steps in the testing process. Even at the unit level, the performance of an individual module may be assessed as tests are conducted. However, it is not until all system elements are fully integrated that the true performance of a system can be ascertained.</p> <p>Deployment testing-software must execute on a variety of platforms and under more than one operating system environment. Deployment testing, sometimes called configuration testing, exercises the software in each environment in which it is to operate. In addition, deployment testing examines all installation procedures and specialized installation software (e.g., "installers") that will be used by customers and all documentation that will be used to introduce the software to end users.</p>
11.	<p>What is Code refactoring? Explain different techniques of refactoring? (<i>Apr 16</i>)</p> <p>Code refactoring is the process of restructuring existing computer code- changing factoring without changing its external behavior. Refactoring improves nonfunctional attributes of the software.</p> <p>It is the process of changing a software system in such a way that it does not always alter the external behavior of the code yet improves its internal structure."</p> <p>Just cleaning up code.:-</p> <p>Contrary to idealized development strategy:</p> <ol style="list-style-type: none"> 1. analysis and design 2. code 3. test <p>At first, code is pretty good but as requirements change or new features are added, the code structure tends to atrophy. Refactoring is the process of fixing a bad or chaotic design.</p> <p>Amounts to moving methods around, creating new methods, adding or deleting classes, ...</p> <p>TJP: Sometimes it means completely redoing the entire code base (i.e., throwing stuff away). Avoid the <i>second system effect</i>!</p> <p>Why refactoring??</p> <p>Improve code structure and design</p> <ul style="list-style-type: none"> • more maintainable • easier to understand • easier to modify • easier to add new features

	<p>Cumulative effect can radically improve design rather than normal slide into decay.</p> <p>Flip-flop code development and refactoring. Only refactor when refactoring--do not add features during refactoring.</p> <p>TJP: kind of like an immune system that constantly grooms the body looking for offensive and intrusive entities.</p> <p>Bad code usually takes more code to do the same thing often because of duplication:</p> <p>When not to refactor?</p> <ul style="list-style-type: none"> • Sometimes you should throw things out and start again. • Sometimes you are too close to a deadline.
12.	<p>i) Why does software testing need extensive planning? Explain. (<i>Apr 16</i>)</p> <p>ii) Compare and contrast alpha and beta testing. (<i>Apr16</i>)</p> <p>Software testing is much harder than coding. You can develop a program easily, but to debug it is a tough task, so it needs extensive planning, because testing is required on the each phase or line of a program, it can be white box testing(line to line) or black box testing(function to function)</p> <p>ii) Compare and contrast alpha and beta testing. (MAY/JUNE 2016)</p> <p>Alpha and Beta Testing It is virtually impossible for a software developer to foresee how the customer will really use a program. Instructions for use may be misinterpreted; strange combinations of data may be regularly used; output that seemed clear to the tester may be unintelligible to a user in the field.</p> <p>When custom software is built for one customer, a series of acceptance tests are conducted to enable the customer to validate all requirements. Conducted by the enduser rather than software engineers, an acceptance test can range from an informal "test drive" to a planned and systematically executed series of tests. In fact, acceptance testing can be conducted over a period of weeks or months, thereby uncovering cumulative errors that might degrade the system over time.</p> <p>If software is developed as a product to be used by many customers, it is impractical to perform formal acceptance tests with each one. Most software product builders use a process called alpha and beta testing to uncover errors that only the end-user seems able to find.</p> <p>The alpha test is conducted at the developer's site by a customer. The software is used in a natural setting with the developer "looking over the shoulder" of the user and recording errors and usage problems. Alpha tests are conducted in a controlled environment.</p> <p>The beta test is conducted at one or more customer sites by the end-user of the software. Unlike alpha testing, the developer is generally not present. Therefore, the beta test is a "live" application of the software in an environment that cannot be controlled by the developer. The customer records all problems (real or imagined) that are encountered during beta testing and reports these to the developer at regular intervals. As a result of problems reported during beta tests, software engineers make modifications and then prepare for release of the software product to the entire customer base.</p>
13.	<p>Consider the pseudo code for simple subtraction given below? (<i>Apr 17, 18</i>)</p> <p>Program 'Simple Subtraction'</p> <p>Input(x,y)</p> <p>Output(x)</p> <p>Output(y)</p>

If $x > y$ then DO

$x - y = z$

Else $y - x = z$

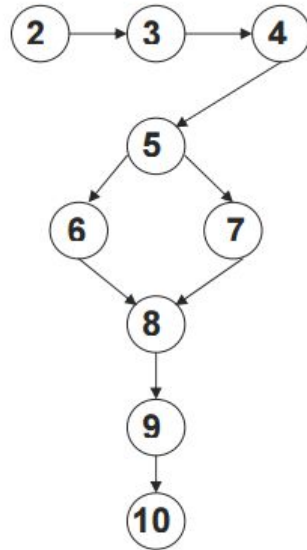
Endif

Output(z)

Output 'End Program'

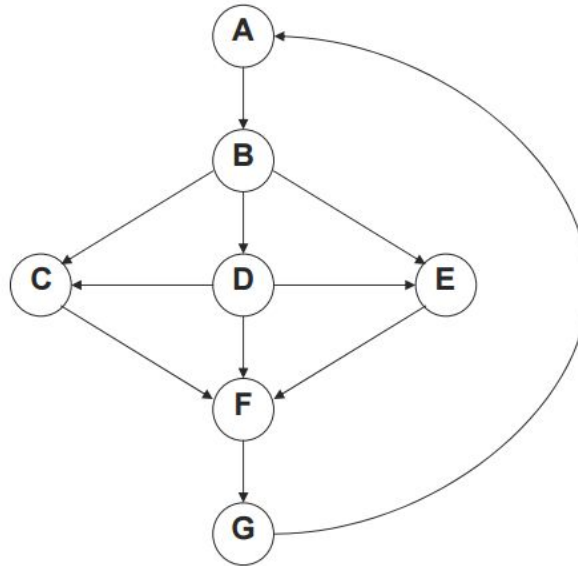
Perform basic path testing and generate test cases.

The construction of a program graph for this simple code is a basic task. Each line number is used to enumerate the relevant nodes of the graph.



Starting at the source node and ending at the sink node, there exist two possible paths. The first path would be the result of the If-Then clause being taken, and the second would be the result of the Else clause being taken. A program graph provides us with some interesting details about the structure of a piece of code. In the example graph given in the above figure, we can see that nodes 2 through to 4 and nodes 9 to 10 are sequences. This means that these nodes represent simple statements such as variable declarations, expressions or basic input/output commands. Nodes 5 through to 8 are a representation of an if then-else construct, while nodes 2 and 10 are the source and sink nodes of the program respectively.

A program may contain thousands of lines of code and remain structured, whereas a piece of code only ten lines long may contain a loop that results in a loss of structure, and thus spores a potentially large number of execution paths. This is shown by the simple program graph given in the figure below.



A simple yet unstructured graph

Basis Path Testing

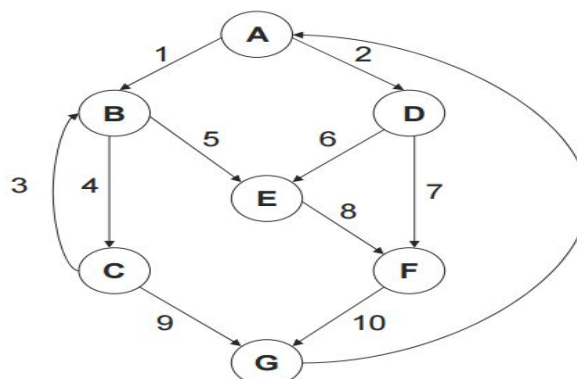
The basis of a vector space contains a set of vectors that are independent of one another, and have a spanning property; this means that everything within the vector space can be expressed in terms of the elements within the basis. The method devised by McCabe to carry out basis path testing has four steps.

These are:

1. Compute the program graph.
2. Calculate the cyclomatic complexity.
3. Select a basis set of paths.
4. Generate test cases for each of these paths

It demonstrates how the basis of a graph containing a loop is computed. It should be noted that the graph is strongly connected; that is, there exists an edge from the sink node to the source node.

Strongly connected program graph

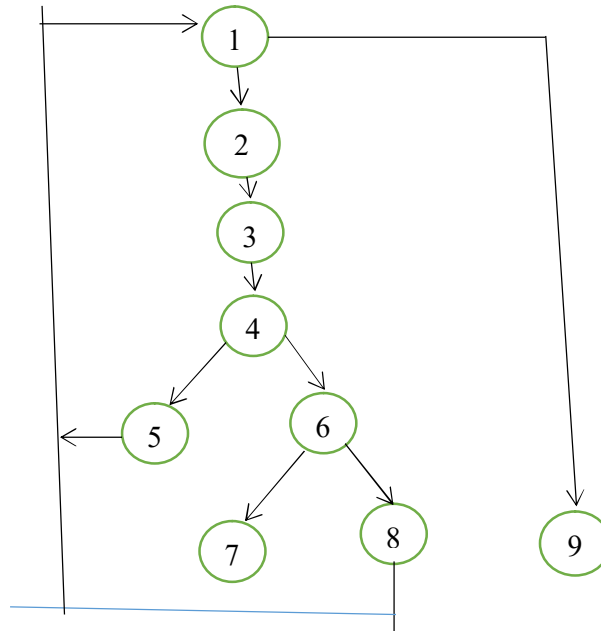


The cyclomatic complexity of a strongly connected graph is provided by the formula

	$V(G) = e - n + p$ $V(G) = e - n + p = 11 - 7 + 1 = 5$ <p>The five linearly independent paths of our graph are as follows:</p> <p>Path 1: A, B, C, G.</p> <p>Path 2: A, B, C, B, C, G.</p> <p>Path 3: A, B, E, F, G.</p> <p>Path 4: A, D, E, F, G.</p> <p>Path 5: A, D, F, G</p> <p>This now forms the basis set of paths for the graph in the above figure. In theory, if we allow for the basic notions of scalar multiplication and addition, we should now be able to construct any path from our basis. Let us attempt to create a 6th path: A, B, C, B, E, F, G. This is the basis sum $p_2 + p_3 - p_1$. This equation means to concatenate paths 2 and 3 together to form the path A, B, C, B, C, G, A, B, E, F, G and then remove the four nodes that appear in path 1, resulting in the required path 6.</p>
14.	<p>Describe black box testing .Design the black box test suite for the following program .The program computes the intersection point of two straight line and display the result .It reads two integer pairs (m1, c1) and (m2,c2) defining the two straight lines of the form $y=mx+c$. (Apr 17)</p> <p>The equivalence classes are the following:</p> <ul style="list-style-type: none"> • Parallel lines ($m_1=m_2, c_1 \neq c_2$) • Intersecting lines ($m_1 \neq m_2$) • Coincident lines ($m_1=m_2, c_1=c_2$) <p>Now, selecting one representative value from each equivalence class, the test suit (2, 2)(2, 5), (5, 5) (7, 7), (10, 10) (10, 10) are obtained.</p> <p>Boundary Value Analysis Boundary value analysis based test suit design involves designing test cases using the values at the boundaries of the different equivalent classes. A type of programming error frequently occurs at the boundaries of different equivalence classes of inputs. The reason behind such errors might purely be due to psychological factors. So in order to design boundary value test cases, it is required to examine the equivalent class to check, if any of equivalence class contain a range of values. Programmers often fail to see the special processing required by the input values that lie at the boundary of the different equivalence classes. For example, programmers may improperly use < instead of <=, or conversely <=for <. Boundary value analysis leads to selection of test cases at the boundaries of the different equivalence classes.</p>
15.	<p>Consider the following program segment.(Nov 17)</p> <pre> /* num is the number the function searches in a presorted integer array arr */ int bin_search (int num) { Int min, max;min=0;max=100; While(min!=max) { If(arr[(min+max)/2]>num) Max = (min + max)/2; Else if(arr[min + max)/2] Min =(min +max)/2; Else return ((min + max)/2); } Return(-1); } </pre>

- i) Draw the control flow graph for this program segment.
- ii) Define cyclomatic complexity.
- iii) Determine the cyclomatic complexity for this program

Cyclomatic complexity is a software metric (measurement), used to indicate the complexity of a program. It is a quantitative measure of the number of linearly independent paths through a program's source code. Cyclomatic complexity is computed using the control flow graph of the program: the nodes of the graph correspond to indivisible groups of commands of a program, and a directed edge connects two nodes if the second command might be executed immediately after the first command. Cyclomatic complexity may also be applied to individual functions, modules, methods or classes within a program.



$$\begin{aligned}
 \text{iii) } V &= E - N + 2 \\
 &= 11 - 9 + 2 \\
 &= 4
 \end{aligned}$$

$$\begin{aligned}
 V(G) &= \text{NO. OF PREICATES} + 1 \\
 &= 3 + 1 \\
 &= 4
 \end{aligned}$$

NO OF PATHS=4

Path1: 1 2 3 4 5 1 9

Path2: 1 2 3 4 6 7 1 9

Path3: 1 2 3 4 6 8 1 9

Path4: 1 9

16. Explain how the various types of loops are tested. (Nov 17, 18)

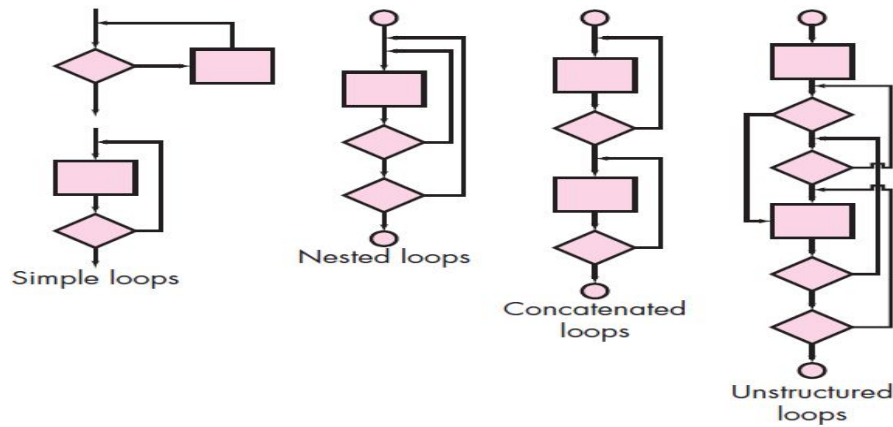
Loop Testing

Loops are the cornerstone for the vast majority of all algorithms implemented in software. And yet, we often pay them little heed while conducting software tests. *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.



4. m passes through the loop where $m \leq n$.

5. $n - 1, n, n + 1$ 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. Beizer suggests an approach that will help to reduce the 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 parameter (e.g., loop counter) values. Add other tests for out-of-range or excluded values.
3. Work outward, 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. When the loops are not independent, the approach applied to nested loops is recommended.

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

17. Differentiate black box testing and white box testing. (Nov 17, 18, Apr 19)

Criteria	Black Box Testing	White Box Testing
Definition	Black Box Testing is a software testing method in which the internal structure/ design/ implementation of the item being tested is NOT known to the tester	White Box Testing is a software testing method in which the internal structure/ design/ implementation of the item being tested is known to the tester.
Levels Applicable To	Mainly applicable to higher levels of testing: Acceptance Testing System Testing	Mainly applicable to lower levels of testing: Unit Testing Integration Testing

	Responsibility	Generally, independent Software Testers	Generally, Software Developers
	Programming Knowledge	Not Required	Required
	Implementation Knowledge	Not Required	Required
	Basis for Test Cases	Requirement Specifications	Detail Design
18.	<p>i) Define : Refactoring (<i>Apr 19</i>)</p> <p>ii) List the phases in software reengineering process model and explain each phase. Planning ,Analysis, Design, Development, Testing and Maintenance</p> <p>(i)Regression testing. (ii)Refactoring (iii)Debugging</p> <p>i) Regression testing. Each time a new module is added as part of integration testing, the software changes. New data flow paths are established, new I/O may occur, and new control logic is invoked. These changes may cause problems with functions that previously worked flawlessly. In the context of an integration test strategy, regression testing is the re execution of some subset of tests that have already been conducted to ensure that changes have not propagated unintended side effects. In a broader context, successful tests (of any kind) result in the discovery of errors, and errors must be corrected. Whenever software is corrected, some aspect of the software configuration (the program, its documentation, or the data that support it) is changed. Regression testing helps to ensure that changes (due to testing or for other reasons) do not introduce unintended behavior or additional errors. Regression testing may be conducted manually, by re executing a subset of all test cases or using automated capture/playback tools. Capture/playback tools enable the software engineer to capture test cases and results for subsequent playback and comparison. The regression test suite (the subset of tests to be executed) contains three different classes of test cases:</p> <ul style="list-style-type: none"> • A representative sample of tests that will exercise all software functions. • Additional tests that focus on software functions that are likely to be affected by the change. • Tests that focus on the software components that have been changed. <p>ii) Refactoring Refactoring is the process of restructuring existing computer code– changing factoring without changing its external behavior. Refactoring improves nonfunctional attributes of the software. It is the process of changing a software system in such a way that it does not always alter the external behavior of the code yet improves its internal structure."</p> <p>Just cleaning up code:-</p> <p>Contrary to idealized development strategy:</p> <ol style="list-style-type: none"> 4. analysis and design 5. code 		

6. test

At first, code is pretty good but as requirements change or new features are added, the code structure tends to atrophy. Refactoring is the process of fixing a bad or chaotic design.

Amounts to moving methods around, creating new methods, adding or deleting classes, ...

TJP: Sometimes it means completely redoing the entire code base (i.e., throwing stuff away).
Avoid the *second system effect*!

Why refactoring??

Improve code structure and design

- more maintainable
- easier to understand
- easier to modify
- easier to add new features

Cumulative effect can radically improve design rather than normal slide into decay.

Flip-flop code development and refactoring. Only refactor when refactoring--do not add features during refactoring.

TJP: kind of like an immune system that constantly grooms the body looking for offensive and intrusive entities.

Bad code usually takes more code to do the same thing often because of duplication:

When not to refactor?

Sometimes you should throw things out and start again.

Sometimes you are too close to a deadline.

iii) Debugging

Debugging occurs as a consequence of successful testing. That is, when a test case uncovers an error, debugging is the process that results in the removal of the error. Although debugging can and should be an orderly process, it is still very much an art. As a software engineer, you are often confronted with a "symptomatic" indication of a software problem as you evaluate the results of a test. That is, the external manifestation of the error and its internal cause may have no obvious relationship to one another. The poorly understood mental process that connects a symptom to a cause is debugging.

Debugging is not testing but often occurs as a consequence of testing.

19. Compare and contrast reverse engineering, forward engineering and reengineering. (Apr 21)

Forward Engineering	Reverse Engineering	Reengineering
In forward engineering, the application are developed with the given requirements.	In reverse engineering or backward engineering, the information are collected from the given application.	Re-engineering is the reorganizing and modifying existing software systems to make

			them more maintainable.	
	Forward Engineering is a high proficiency skill.	Reverse Engineering or backward engineering is a low proficiency skill.	The extent of the required data conversion	
	Forward Engineering takes more time to develop an application.	While Reverse Engineering or backward engineering takes less time to develop an application.	The availability of expert staff for re-engineering	
	The nature of forward engineering is Prescriptive.	The nature of reverse engineering or backward engineering is Adaptive.	The quality of the software to be re-engineered	
	In forward engineering, production is started with given requirements.	In reverse engineering, production is started by taking the products existing products.	Better use of Existing Staff	
	The example of forward engineering is the construction of electronic kit, construction of DC MOTOR , etc.	An example of backward engineering is research on Instruments etc.	The tool support available for re-engineering	
20.	Write a procedure for the following : Given three sides of a triangle, return the type of triangle i.e. equilateral, isosceles and scalene triangle. Draw the Control Flow Graph and calculate cyclomatic complexity to calculate the minimum number of paths. Enumerate the paths to be tested. (Apr 19)			
21.	Write notes on: (i)Regression testing. (ii)Refactoring (iii)Debugging			
UNIT V PROJECT MANAGEMENT				
Software Project Management: Estimation - LOC, FP Based Estimation, Make/Buy Decision COCOMO I & II Model - Project Scheduling - Scheduling, Earned Value Analysis Planning - Project Plan, Planning Process, RFP Risk Management - Identification, Projection - Risk Management-Risk Identification-RMMM Plan-CASE TOOLS.				
UNIT V/PART-A				
1.	What is project planning. (Apr 12, 15) The objective of project planning is to provide a framework that enables the manager to make reasonable estimates of resources, cost and schedule.			
2.	What is risk management? (Nov 16) Risk management is a series of steps that helps a software team to understand and manage uncertainty.			
3.	What are the processes of risk management? (Nov 10, Apr 21) <ul style="list-style-type: none"> • Risk identification • Risk analysis • Risk Monitoring 			
4.	Give the procedure of the Delphi method.			

	<ul style="list-style-type: none"> • The coordinator presents a specification and estimation form to each expert. • Coordinator calls a group meeting in which the experts discuss estimation issues with the coordinator and each other. • Experts fill out forms anonymously. • Coordinator prepares and distributes a summary of the estimates. • The Coordinator then calls a group meeting. In this meeting the experts mainly discuss the points where their estimates vary widely. • The experts again fill out forms anonymously. <p>Again coordinator edits and summarizes the forms and the steps 5 and 6 are followed until the coordinator is satisfied with the overall prediction synthesized from experts.</p>
5.	<p>How is the accuracy of a software project is estimated?</p> <ul style="list-style-type: none"> • The degree to which the planner has properly estimated the size of the product to be built. • The ability to translate the size estimate to human effort. • The degree to which the project plan reflects the abilities of the software team. <p>The stability of the product requirements and the environment that supports the software engineering effort.</p>
6.	<p>What is Post-architecture stage model?</p> <p>Post-architecture stage model is used during the construction of the software.</p>
7.	<p>What is Risk? Give an example of risk (Nov 13 , 14)</p> <p>Risk concerns with failure happenings. Risk involves change, such as in changes in mind, opinion, actions, or places. Risk involves choice and the uncertainty that choice itself entails. A risk is a potential problem—it might happen, it might not. But, regardless of the outcome, it's a really good idea to identify it, assess its probability of occurrence, estimate its impact, and establish a contingency plan should the problem actually occur.</p> <p>Example: Unrealistic delivery rate, Staff turnover</p>
8.	<p>Give the interpretation of software metrics?</p> <ul style="list-style-type: none"> • Number derived from a software process or product of resource. • Scale of measurement identifiable software attributes. • Data- driven model.
9.	<p>What is application composition model?</p> <p>Application Composition model uses object points, an indirect software measure that is computed using user interfaces, reports and components.</p>
10.	<p>Define Software Quality (Nov 13)</p> <p>Software quality is a complex mix of factors that will vary across different applications and the customers who request them</p>
11.	<p>What are different risk strategies?</p> <ul style="list-style-type: none"> • Reactive risk strategy-monitors the project for likely risks. <p>Proactive risk strategy-Potential risks are identified, their probability and impact are assessed, and they are ranked by importance.</p>
12.	<p>Define software measure. Give the 5 characteristics of the software measurement.</p> <p>A Software measure is a mapping from a set of objects in the software engineering world into a set of mathematical constructs such as numbers or vectors of numbers.</p>

	<ul style="list-style-type: none"> Object of measurement, Purpose of measurement. Source of measurement, Measured property, Context of measurement.
13.	What are the different sizing options? <ul style="list-style-type: none"> Object points , Function points , Lines of Source code.
14.	What is called software maintenance? The maintenance of existing software can account for over 60% of all effect expend by a development organization, and the percentage continues to rise as more software is produced.
15.	What is Risk exposure? The risk exposure is determined by, $RE=P*C$ where P is the probability of occurrence for a risk and C is the cost to the project should the risk occur.
16.	What are the various cost estimation techniques? <ul style="list-style-type: none"> Problem-based estimation , LOC-based estimation. FP-based estimation , Process-based estimation.
17.	What is meant by Process metrics? Process metrics assess the effectiveness and quality of software process, determine maturity of the process, effort required in the process , effectiveness of defect removal during development and so on.
18.	How are the software risks assessed? (Nov 12) By answering the questions have derived from risk data obtained by surveying experienced software project managers in different part of the world . The questions are ordered by their relative importance to the success of a project. If any one of these questions is answered negatively, mitigation, monitoring, and management steps should be instituted without fail.
19.	What are the methods of cost estimation? <ul style="list-style-type: none"> Constructive cost model and Delphi method of cost estimation.
20.	What are the three different types of software maintenance? <ul style="list-style-type: none"> Perfective maintenance or enhancement , Preventive maintenance or reengineering. Corrective maintenance , Adaptive maintenance.
21.	What is Early stage design model? Early stage design model is used once requirements have been stabilized and the basic software architecture has been established.
22.	What is EVA? (Apr 18) Earned Value Analysis is a technique of performing quantitative analysis of the software project. It provides a common value scale for every task of software project. It acts as a measure for software project progress.
23.	List a few process and project metrics. (Apr 16) <u>Project metrics:</u> Schedule Variance, Effort Variance, Size Variance, Requirement Stability Index, Productivity, Schedule variance for a phase, Effort variance for a phase. <u>Process Metrics:</u> Cost of quality, Cost of poor quality, Defect density, Review efficiency, Testing Efficiency, Defect removal efficiency.
24.	Will exhaustive testing guarantee that the program is 100% correct? (Apr 16) No, even exhaustive testing will not guarantee that the program is 100 percent correct.
25.	Enumerate the factors that influence a project schedule. (Nov 18)

	a) Smart people b) Smart Planning c) Open Communication d) Careful Risk Management.
26.	<p>List two customer related and technology related risks.(Apr 17)</p> <p>Customer related risks</p> <ul style="list-style-type: none"> • Is the customer technically sophisticated in the product area • Does the customer have a solid idea of what is required • Is the customer willing to establish rapid communication links with the developer <p>Technology related risks</p> <ul style="list-style-type: none"> • Is the technology to be built new to your company • Is a specialized user interface demanded by product requirements
27.	<p>What is COCOMO model?</p> <p>CONstructive COSt MOdel is a cost model, which gives the estimate of number of man-months it will take to develop the software product.</p>
28.	<p>Write a note on Risk information sheet(RIS) (Nov 18)</p> <p>Some software teams do not develop a formal RMMM document. Rather, each risk is documented individually using a <i>risk information sheet</i> (RIS) In most cases, the RIS is maintained using a database system so that creation and information entry, priority ordering, searches, and other analysis may be accomplished easily.</p>
29.	<p>How is productivity and cost related to function points?(Nov 16)</p> <p>Productivity:</p> <p>The definition of productivity is the output-input ratio within a time period with due consideration for quality.</p> <p>Productivity = outputs/inputs (within a time period, quality considered)</p> <p>The formula indicates that productivity can be improved by</p> <ol style="list-style-type: none"> (1) by increasing outputs with the same inputs, (2) by decreasing inputs but maintaining the same outputs, or (3) by increasing outputs and decreasing inputs change the ratio favorably. <p>Software Productivity = Function Points / Inputs</p> <p>Software productivity is defined as hours/function points or function points/hours. This is the average cost to develop software or the unit cost of software. Average cost is the total cost of producing a particular quantity of output divided by that quantity i.e., Total Cost/Function Points.</p>
30.	<p>What are the different types of productivity estimation measures?(Apr 17)</p> <p>LOC and FP measures are often used to derive productivity metrics. LOC and FP based metrics have been found to be relatively accurate predictors of software development effort and cost.</p>
31.	<p>List the case tools for the following phases of SDLC : Design, Testing(Apr 19)</p> <p>Upper Case Tools - This tools are used in planning, analysis and design stages of SDLC.</p> <p>Lower Case Tools - Lower CASE tools are used in implementation, testing and maintenance.</p> <p>Integrated Case Tools - it helps in all the stages of SDLC, from Requirement gathering to Testing and documentation.</p>
32.	<p>Compare Project Risk vs Business Risk (Apr 19)</p> <p>The important distinction that must be understood is the difference between business risks and project risks. Business risks are more general and relate to the organization, whereas project risks relate specifically to the project objectives.</p>
33.	<p>list out the various steps in planning process. (Apr 21)</p>

	<div>1. Perception of Opportunities</div> <div>2. Establishing Objectives</div> <div>3. Planning Premises</div> <div>4. Identification of Alternatives</div> <div>5. Evaluation of Alternatives</div> <div>6. Choice of Alternative Plans</div> <div>7. Formulation of Supporting Plan</div> <div>8. Establishing Sequence of Activities.</div>																			
34.	<p>List two advantages of using COCOMO model (Apr 19)</p> <ul style="list-style-type: none">• COCOMO is factual and easy to interpret. One can clearly understand how it works.• Accounts for various factors that affect cost of the project.• Works on historical data and hence is more predictable and accurate. <p>The drivers are very helpful to understand the impact on the different factors that affect the project costs.</p>																			
UNIT-V / PART -B																				
1.	<p>(i) Explain the COCOMO II model for estimation. (ii) Make/Buy Decision.</p> <p>(iii)What is the purpose of Delphi method? State advantages and disadvantages of the method. (Nov 12, 13, 17, Apr 16, 17, 18, 21)</p> <p>Barry Boehm introduced COCOMO II (COst COnstructive MOdel) which is an hierarchy of estimation models that addresses the following areas:</p> <ul style="list-style-type: none">• Application composition model. Used during the early stages of software engineering, when prototyping of user interfaces, consideration of software and system interaction, assessment of performance, and evaluation of technology maturity are paramount.• 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. <p>Three different sizing options are available as part of the model hierarchy: object points, function points, and lines of source code.</p> <table><tr><th rowspan="2">OBJECT TYPE</th><th colspan="3">COMPLEXITY WEIGHT</th></tr><tr><th>SIMPLE</th><th>MEDIUM</th><th>DIFFICULT</th></tr><tr><td>Screen</td><td>1</td><td>2</td><td>3</td></tr><tr><td>Report</td><td>2</td><td>5</td><td>8</td></tr><tr><td>3GL component</td><td></td><td></td><td>10</td></tr></table> <p>The object point is an indirect software measure that is computed using counts of the number of (1) screens (at the user interface), (2) reports, and (3) components likely to be required to build the application.</p> <p>When component-based development or general software reuse is to be applied, the percent of reuse (%reuse) is estimated and the object point count is adjusted:</p> $NOP=(Object\ points)*[(100-\%reuse)/100]$ <p>where NOP is defined as new object points.</p> <p>To derive an estimate of effort based on the computed NOP value, a “productivity rate” must be derived.</p> $PROD = \frac{NOP}{Person-month}$ <p>for different levels of developer experience and development environment maturity.</p> <p>Once the productivity rate has been determined, an estimate of project effort is computed using</p>	OBJECT TYPE	COMPLEXITY WEIGHT			SIMPLE	MEDIUM	DIFFICULT	Screen	1	2	3	Report	2	5	8	3GL component			10
OBJECT TYPE	COMPLEXITY WEIGHT																			
	SIMPLE	MEDIUM	DIFFICULT																	
Screen	1	2	3																	
Report	2	5	8																	
3GL component			10																	

NOP

Estimated effort=-----

PROD

In more advanced COCOMO II models,12 a variety of scale factors, cost drivers, and adjustment procedures are required.

Developer's experience/capability	Very low	Low	Nominal	High	Very high
Environment maturity/capability	Very low	Low	Nominal	High	Very high
PROD	4	7	13	25	50

(ii)State ZIPF's law.

Frequency of occurrence of words is inversely proportional to the rank in this frequency of occurrence

(iii)What is the purpose of Delphi method? State advantages and disadvantages of the method. **(NOV/DEC 2014)**

Under this method of software estimation, the project specifications would be given to a few experts and their opinion taken. The actual number of experts chosen would depend on their availability. A minimum of three is normally selected to have a range of values. Delphi method has the following steps -

- Selection of experts
- Briefing to the experts
- Collation of estimates from experts
- Convergence of estimates and finalization

Merits of Delphi technique

- Very useful when the organization does not have any in-house experts with the domain knowledge or the development platform experience to come out with a quick estimate
- Very quick to derive an estimate
- Simple to administer and use

Demerits of Delphi technique

- This is too simplistic
- It may be difficult to locate right experts
- It may also be difficult to locate adequate number of experts willing to participate in the estimation
- The derived estimate is not auditable
- It is not possible to determine the causes of variance between the estimated value and the actual values
- Only size and effort and estimation are possible - schedule would not be available.

2.

Describe in detail about project scheduling. *(Apr 13, 15)*

Software project scheduling is an action that distributes estimated effort across the planned project duration by allocating the effort to specific software engineering tasks. Scheduling for software engineering projects can be viewed from two rather different perspectives. In the first, an end date for release of a computer-based system has already (and irrevocably) been established. The software organization is constrained to distribute effort within the prescribed

time frame. The second view of software scheduling assumes that rough chronological bounds have been discussed but that the end date is set by the software engineering organization.

Basic Principles

- **Compartmentalization.** The project must be compartmentalized into a number of manageable activities and tasks. To accomplish compartmentalization, both the product and the process are refined.
- **Interdependency.** The interdependency of each compartmentalized activity or task must be determined. Some tasks must occur in sequence, while others can occur in parallel. Some activities cannot commence until the work product produced by another is available. Other activities can occur independently.
- **Time allocation.** Each task to be scheduled must be allocated some number of work units (e.g., person-days of effort). In addition, each task must be assigned a start date and a completion date that are a function of the interdependencies and whether work will be conducted on a full-time or part-time basis.
- **Effort validation.** Every project has a defined number of people on the software team. As time allocation occurs, you must ensure that no more than the allocated number of people has been scheduled at any given time. For example, consider a project that has three assigned software engineers (e.g., three person-days are available per day of assigned effort). On a given day, seven concurrent tasks must be accomplished. Each task requires 0.50 person-days of effort. More effort has been allocated than there are people to do the work.
- **Defined responsibilities.** Every task that is scheduled should be assigned to a specific team member.
- **Defined outcomes.** Every task that is scheduled should have a defined outcome. For software projects, the outcome is normally a work product (e.g., the design of a component) or a part of a work product. Work products are often combined in deliverables.
- **Defined milestones.** Every task or group of tasks should be associated with a project milestone. A milestone is accomplished when one or more work products has been reviewed for quality.

The Relationship Between People and Effort

L , is related to effort and development time by the equation:

$$L = P \cdot E^{1/3} t^{4/3}$$

Rearranging this software equation, we can arrive at an expression for development effort E :

$$E = \frac{L^3}{P^3 t^4}$$

Effort Distribution

A recommended distribution of effort across the software process is often referred to as the 40–20–40 rule. Forty percent of all effort is allocated to frontend analysis and design. A similar percentage is applied to back-end testing. 20 percent of effort is deemphasized in the coding. Because of the effort applied to software design, code should follow with relatively little difficulty. A range of 15 to 20 percent of overall effort can be achieved. Testing and subsequent debugging can account for 30 to 40 percent of software development effort. The criticality of the software often dictates the amount of testing that is required. If software is human rated (i.e., software failure can result in loss of life), even higher percentages are typical.

3.	<p>Brief about calculating Earned Value Measures. (Apr 12)</p> <p>The earned value system provides a common value scale for every [software project] task, regardless of the type of work being performed. The total hours to do the whole project are estimated, and every task is given an earned value based on its estimated percentage of the total.</p> <p>To determine the earned value, the following steps are performed:</p> <ul style="list-style-type: none"> • The budgeted cost of work scheduled (BCWS) is determined for each work task represented in the schedule. • The BCWS values for all work tasks are summed to derive the budget at completion (BAC). Hence, $BAC = \sum(BCWS_k) \text{ for all task } k$ <ul style="list-style-type: none"> • Next, the value for budgeted cost of work performed (BCWP) is computed. The value for BCWP is the sum of the BCWS values for all work tasks that have actually been completed by a point in time on the project schedule. <p>Given values for BCWS, BAC, and BCWP, important progress indicators can be computed:</p> $\text{Schedule performance index, SPI} = \frac{BCWP}{BCWS}$ $\text{Schedule Variance, SV} = BCWP - BCWS$ $\text{Percent scheduled for completion} = \frac{BCWP}{BAC}$ $\text{Percent complete} = \frac{BCWP}{BAC}$ $\text{Cost performance index, CPI} = \frac{BCWP}{ACWP}$ $\text{Cost variance, CV} = BCWP - ACWP$ <p>A CPI value close to 1.0 provides a strong indication that the project is within its defined budget. CV is an absolute indication of cost savings (against planned costs) or shortfall at a particular stage of a project. Like over-the-horizon radar, earned value analysis illuminates scheduling difficulties before they might otherwise be apparent. This enables you to take corrective action before a project crisis develops.</p>
4.	<p>Define Risk. Explain its four major categories. (OR) State the need for Risk Management and explain the activities under Risk Management. (Apr 15, 19, 21, Nov 17)</p> <p>Risk concerns with failure happenings. Risk involves change, such as in changes in mind, opinion, actions, or places.</p> <ul style="list-style-type: none"> • uncertainty – the risk may or may not happen; • loss – if the risk becomes a reality, unwanted consequences or losses will occur <p>Project risks threaten the project plan. That is, if project risks become real, it is likely that the project schedule will slip and that costs will increase. Project risks identify potential budgetary, schedule, personnel (staffing and organization), resource, stakeholder, and requirements problems and their impact on a software project.</p> <p>Technical risks threaten the quality and timeliness of the software to be produced. If a technical risk becomes a reality, implementation may become difficult or impossible. Technical risks identify potential design, implementation, interface, verification, and maintenance</p>

problems.

Business risks threaten the viability of the software to be built and often jeopardize the project or the product. Candidates for the top five business risks are

- building an excellent product or system that no one really wants (market risk),
- building a product that no longer fits into the overall business strategy for the company (strategic risk),
- building a product that the sales force doesn't understand how to sell (sales risk),
- losing the support of senior management due to a change in focus or a change in people (management risk), and
- losing budgetary or personnel commitment (budget risks).

Known risks are those that can be uncovered after careful evaluation of the project plan, the business and technical environment in which the project is being developed, and other reliable information sources (e.g., unrealistic delivery date, lack of documented requirements or software scope, poor development environment).

Predictable risks are extrapolated from past project experience (e.g., staff turnover, poor communication with the customer, dilution of staff effort as ongoing maintenance requests are serviced). Unpredictable risks are the joker in the deck. They can and do occur, but they are extremely difficult to identify in advance.

Risk Identification

Risk identification is a systematic attempt to specify threats to the project plan (estimates, schedule, resource loading, etc.). By identifying known and predictable risks, the project manager takes a first step toward avoiding them when possible and controlling them when necessary.

There are two distinct types of risks for each of the categories: generic risks and product-specific risks.

- Generic risks are a potential threat to every software project.
- Product-specific risks can be identified only by those with a clear understanding of the technology, the people, and the environment that is specific to the software that is to be built.

One method for identifying risks is to create a risk item checklist. The checklist can be used for risk identification and focuses on some subset of known and predictable risks in the following generic subcategories:

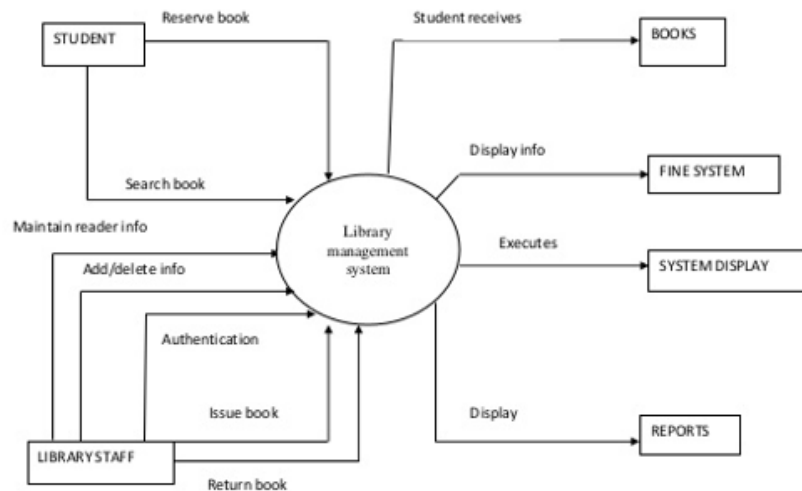
- Product size—risks associated with the overall size of the software to be built or modified.
- Business impact—risks associated with constraints imposed by management or the marketplace.
- Stakeholder characteristics—risks associated with the sophistication of the stakeholders and the developer's ability to communicate with stakeholders in a timely manner.
- Process definition—risks associated with the degree to which the software process has been defined and is followed by the development organization.
- Development environment—risks associated with the availability and quality of the tools to be used to build the product.
- Technology to be built—risks associated with the complexity of the system to be built and the "newness" of the technology that is packaged by the system.
- Staff size and experience—risks associated with the overall technical and project experience of the software engineers who will do the work.

Risk Projection

Risk projection, also called risk estimation, attempts to rate each risk in two ways— the

	<p>likelihood or probability that the risk is real and the consequences of the problems associated with the risk, should it occur. You work along with other managers and technical staff to perform four risk projection steps:</p> <ul style="list-style-type: none"> • Establish a scale that reflects the perceived likelihood of a risk. • Delineate the consequences of the risk. • Estimate the impact of the risk on the project and the product. • Assess the overall accuracy of the risk projection so that there will be no misunderstandings. <p>A risk table provides you with a simple technique for risk projection.</p> <p>The overall risk exposure RE is determined using the following relationship:</p> $RE = P * C$ <p>Risk Refinement</p> <p>One way to do this is to represent the risk in condition-transition-consequence (CTC) format. That is, the risk is stated in the following form:</p> <p style="padding-left: 40px;">Given that <condition> then there is concern that (possibly) <consequence>.</p> <p>This general condition can be refined in the following manner:</p> <p>Sub condition 1. Certain reusable components were developed by a third party with no knowledge of internal design standards.</p> <p>Sub condition 2. The design standard for component interfaces has not been solidified and may not conform to certain existing reusable components.</p> <p>Sub condition 3. Certain reusable components have been implemented in a language that is not supported on the target environment.</p> <p>Risk Mitigation, Monitoring and Management</p> <p>An effective strategy must consider three issues: risk avoidance, risk monitoring, and risk management and contingency planning.</p> <p>To mitigate this risk, a strategy has to be developed for reducing turnover. Among the possible steps to be taken are:</p> <ul style="list-style-type: none"> • Meet with current staff to determine causes for turnover (e.g., poor working conditions, low pay, competitive job market). • Mitigate those causes that are under your control before the project starts. • Once the project commences, assume turnover will occur and develop techniques to ensure continuity when people leave. • Organize project teams so that information about each development activity is widely dispersed. • Define work product standards and establish mechanisms to be sure that all models and documents are developed in a timely manner. • Conduct peer reviews of all work (so that more than one person is “up to speed”). <p>Assign a backup staff member for every critical technologist.</p>
5.	<p>Model a data flow diagram for a “Library Management System”. state functional requirement you are considering. (Nov 17)</p>

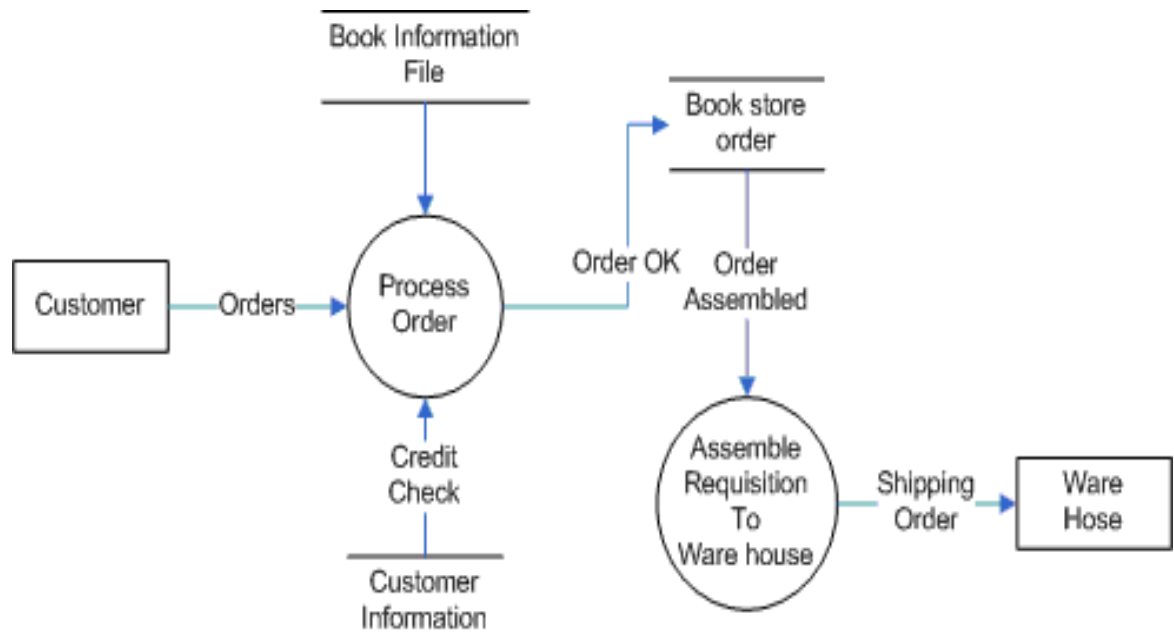
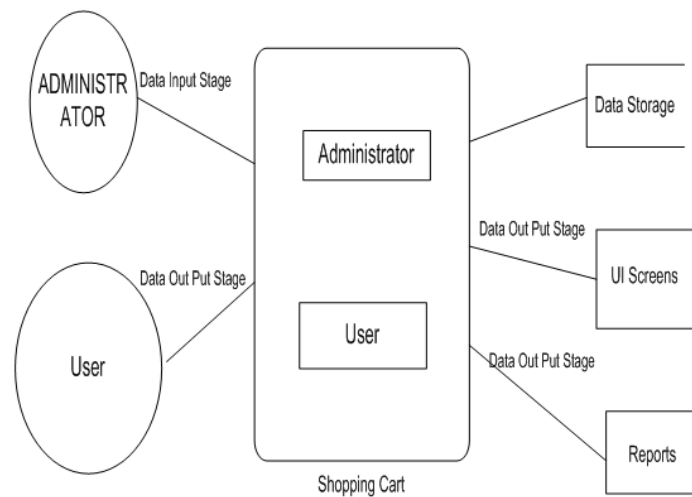
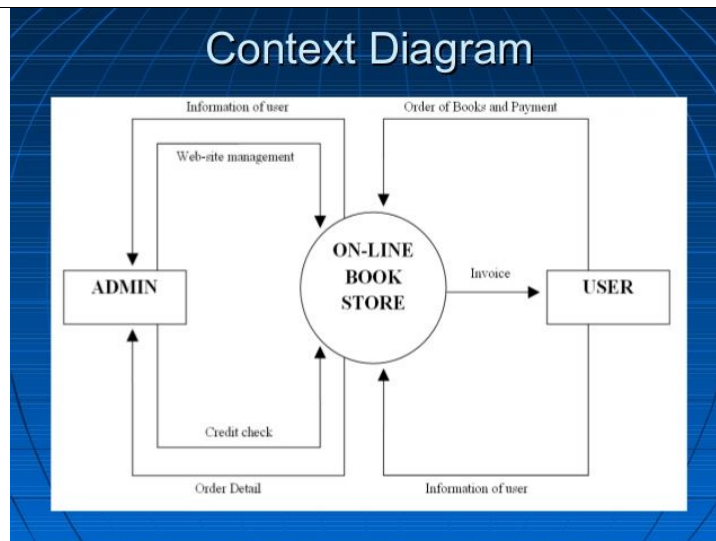
CONTEXT LEVEL DIAGRAM

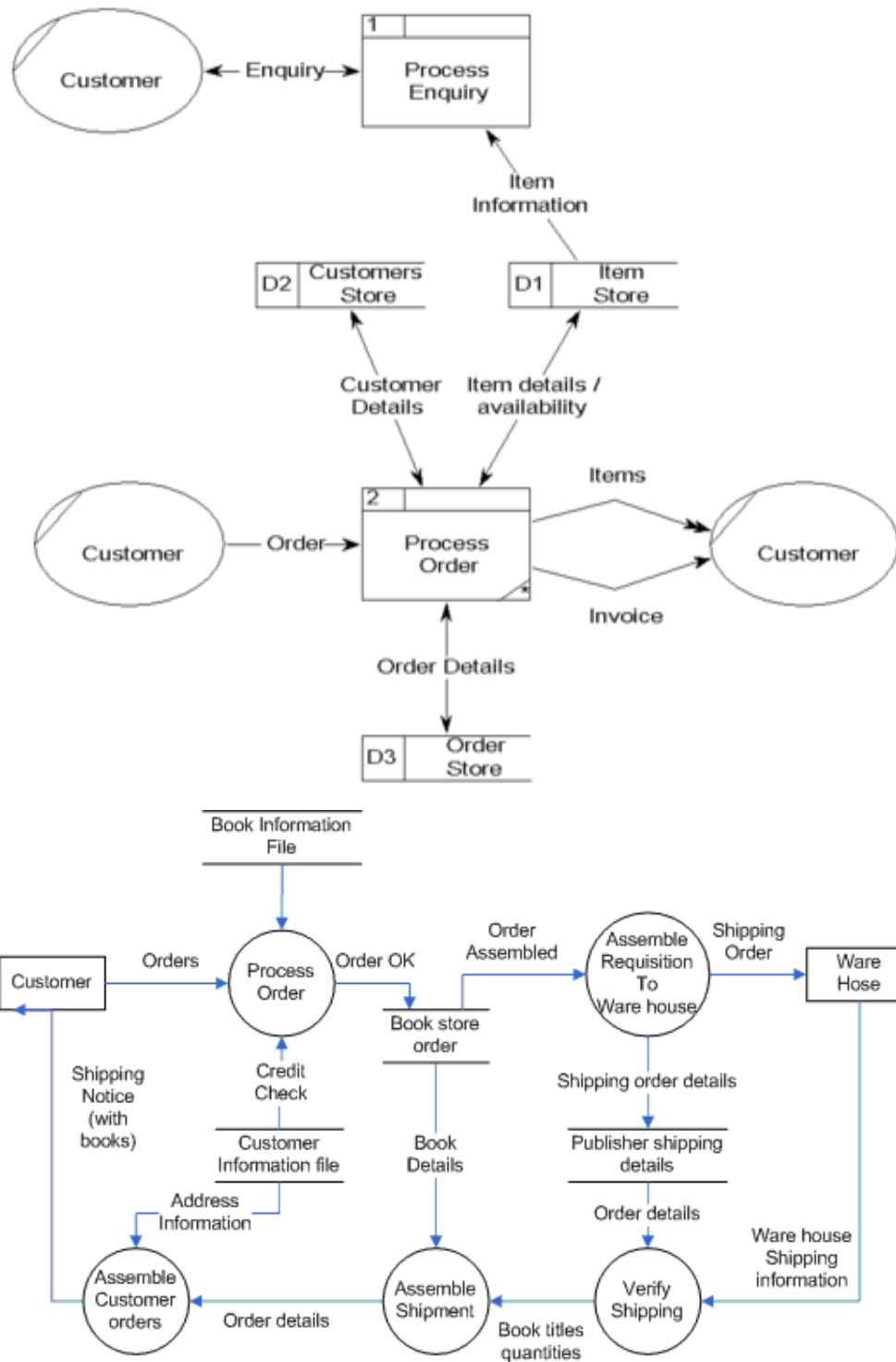


6. Consider an online book stores. It accepts individual/bulk orders, process payments, triggers delivery of the books .some of the major features of the system include: the components of DFD.(Apr 17)

- Order books
- User friendly online shopping cart function
- Create ,view, modify and delete books to be sold
- To store inventory and sales information in database
- To provide an efficient inventory system
- Register for book payment options
- Add a wish list
- Place request for books not available
- To be able to print invoices to members and print a set of summary reports
- Internet access

Analyse the system using the context diagram and level 1 DFD for the system .Explain the components of DFD.(APR/MAY 17)





7. Explain how effort and cost estimation are determined using COCOMO model (Apr 21)
- Application composition model. Used during the early stages of software engineering, when prototyping of user interfaces, consideration of software and system interaction, assessment of performance, and evaluation of technology maturity are paramount.
 - 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.
8. i) An application has the following: 10 low external inputs, 8 high external outputs, 13 low

internal logical files , 17 high external interface files, 11 average external inquiries and complexity adjustment factor of 1.10. What are the unadjusted and adjusted function point counts? (Apr 16)

ii) Discuss Putnam resources allocation model. Derive the time and effort equations.

Measurement Parameter	count	Weighting Factor			Total
		Simple	Average	Complex	
External inputs	10	3	4	6	30
External outputs	8	4	5	6	56
internal logical files	13	7	10	15	91
external interface files	17	5	7	10	170
External inquiries	11	3	4	6	44
					391

$$FP = \text{count total} * [0.65 + 0.01 * \sum fi]$$

Adjustment factor given, $\sum fi = 1.10$

So, Adjusted function points = $391 * [1.75]$

= 684.25 adjusted function points.

9. If team A found 342 errors prior to release of software and Team B found 182 errors. What additional measures and metrics are needed to find out if the teams have removed the errors effectively? Explain.

$$DRE = E / (E + D)$$

E- Error found before release

D- No. of Defect after release

$$DRE = 342 / (342 + 182) = 342 / 524 = 0.65$$

Since DRE is not equal to 1 there exists some defect in the software. DRE can also be used within the project to assess a team's ability to find errors before they are passed to the next framework activity or software engineering task. For example, the requirements analysis task produces an analysis model that can be reviewed to find and correct errors. Those errors that are not found during the review of the analysis model are passed on to the design task (where they may or may not be found). When used in this context, we redefine DRE as

$$DRE_i = E_i / (E_i + E_{i+1})$$

where E_i is the number of errors found during software engineering activity i and E_{i+1} is the number of errors found during software engineering activity $i+1$ that are traceable to errors that were not discovered in software engineering activity i . A quality objective for a software team (or an individual software engineer) is to achieve DRE_i that approaches 1. That is, errors should be filtered out before they are passed on to the next activity.

Measures are:

All the errors are categorized and forwarded to respective team. Internal and external product metrics are measured based on the category of the error.

Example Metrics:

Defects per KLOC
 Errors per FP or Defects per FP
 Defect Density
 Mean time to failure
 Defect density
 Customer problems
 Customer satisfaction

10. Discuss the process of function point analysis. Explain function point analysis with sample cases for components of different complexity.
 Decomposition for FP-based estimation focuses on information domain values rather than software functions, you would estimate inputs, outputs, inquiries, files, and external interfaces for the CAD software. For the purposes of this estimate, the complexity weighting factor is assumed to be average.

Information domain value	Opt.	Likely	Pess.	Est. count	Weight	FP count
Number of external inputs	20	24	30	24	4	97
Number of external outputs	12	15	22	16	5	78
Number of external inquiries	16	22	28	22	5	88
Number of internal logical files	4	4	5	4	10	42
Number of external interface files	2	2	3	2	7	15
Count total						320

Each of the complexity weighting factors is estimated, and the value adjustment factor is computed as below.

Factor	Value
Backup and recovery	4
Data communications	2
Distributed processing	0
Performance critical	4
Existing operating environment	3
Online data entry	4
Input transaction over multiple screens	5
Master files updated online	3
Information domain values complex	5
Internal processing complex	5
Code designed for reuse	4
Conversion/installation in design	3
Multiple installations	5
Application designed for change	5
Value adjustment factor	1.17

Finally, the estimated number of FP is derived: $FP_{estimated} = \frac{Count_{total}}{VAF} = \frac{320}{1.17} = 273.49$. The organizational average productivity for systems of this type is 6.5 FP/pm. Based on a burdened labor rate of \$8000 per month, the cost per FP is approximately \$1230. Based on the FP estimate and the historical productivity data, the total estimated project cost is \$461,000 and the estimated effort is 58 person-months. The LOC and FP estimation techniques differ in the level

	<p>of detail required for decomposition and the target of the partitioning. When LOC is used as the estimation variable, decomposition is absolutely essential and is often taken to considerable levels of detail. The greater the degree of partitioning, the more likely reasonably accurate estimates of LOC can be developed. For FP estimates, decomposition works differently. Rather than focusing on function, each of the information domain characteristics—inputs, outputs, data files, inquiries, and external interfaces—as well as the 14 complexity adjustment values discussed in Chapter 23 are estimated. The resultant estimates can then be used to derive an FP value that can be tied to past data and used to generate an estimate.</p>
11.	<p>i) Discuss about various factors that affects the project plan. (3) 1. Deadline 2. Budget 3. Stakeholders 4. Project Members 5. Demand 6. Supply 7. Price</p> <p>ii) Explain the steps involved in project planning. (10) (Nov 18)</p> <p>Step1: Identify & Meet with Stakeholders Step2: Set & Prioritize Goals Step3: Define Deliverables Step4: Create the Project Schedule Step5: Identify Issues and Complete a Risk Assessment Step6: Present the Project Plan to Stakeholders</p>
12.	<p>List the features of LOC and FP based estimation models. Compare the two models and list the advantage of one over other. (Apr 19)</p> <p>Size-Oriented Metrics</p> <ul style="list-style-type: none"> • LOC measure claim that LOC is an “artifact” of all software development projects that can be easily counted, that many existing software estimation models use LOC or KLOC as a key input, and that a large body of literature and data predicated on LOC already exists. • The planner must estimate the LOC to be produced long before analysis and design has been completed. <p>Function-Oriented Metrics</p> <ul style="list-style-type: none"> • Function-oriented software metrics use a measure of the functionality delivered by the application as a normalization value. The most widely used function-oriented metric is the function point (FP). Computation of the function point is based on characteristics of the software’s information domain and complexity. • The function point, like the LOC measure, is controversial. Proponents claim that FP is programming language independent, making it ideal for applications using conventional and nonprocedural languages, and that it is based on data that are more likely to be known early in the evolution of a project, making FP more attractive as an estimation approach. • Opponents claim that the method requires some “sleight of hand” in that computation is based on subjective rather than objective data, that counts of the information domain (and other dimensions) can be difficult to collect after the fact, and that FP has no direct physical meaning—it’s just a number. <p>Reconciling LOC and FP Metrics</p> <p>The relationship between lines of code and function points depends upon the programming language that is used to implement the software and the quality of the design. Function points and LOC-based metrics have been found to be relatively accurate predictors of software development effort and cost. However, in order to use LOC and FP for estimation, an historical baseline of information must be established.</p> <p>Within the context of process and project metrics, productivity and quality should be</p>

concerned which are the measures of software development “output” as a function of effort and time applied and measures of the “fitness for use” of the work products that are produced. For process improvement and project planning purposes, the interest is historical.

13. Given the requirements for an Automated Teller Machine (ATM) system, Design the following
i) Use case diagram ii) Activity diagram detailing each use case iii) List test cases for any one functionality from your use case diagram. (*Apr 19*)

The ATM will service one customer at a time. A customer will be required to insert an ATM card and enter a personal identification number (PIN)- both of which will be sent to the bank for validation as part of each transaction. The customer will then be able to perform one or more transactions.

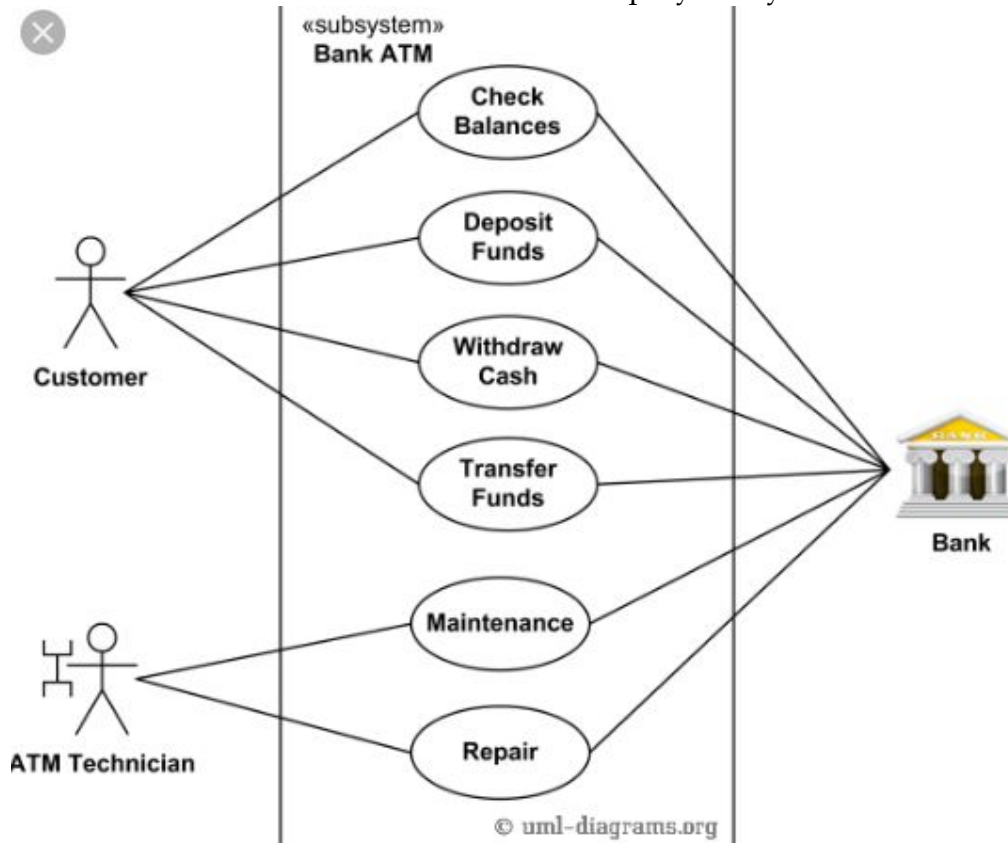
The ATM must be able to provide the following services to the customer:

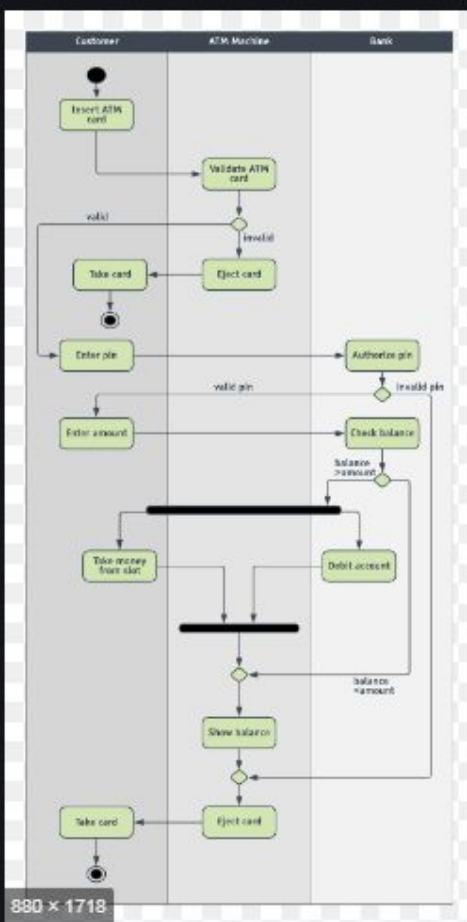
A customer must be able to make a cash withdrawal from any suitable account linked to the card, in multiples of \$20.00. Approval must be obtained from the bank before cash is dispensed.

A customer must be able to make a deposit to any account linked to the card, consisting of cash and/or checks in an envelope. The customer will enter the amount of the deposit into the ATM, subject to manual verification when the envelope is removed from the machine by an operator. Approval must be obtained from the bank before physically accepting the envelope.

A customer must be able to make a transfer of money between any two accounts linked to the card.

A customer must be able to make a balance inquiry of any account linked to the card.





14. Given the following project plan of table1 and table 2: (Apr 17)

ID	Task	Immediate predecessor(*)	Expected Duration	Budget
A	Meet with client		5	500
B	Write SW	A	20	10000
C	Debug SW	B	5	1500
D	Prepare draft manual	B	5	1000
E	Meet with clients	D	5	1000
F	Test SW	C,E	20	2000
G	Make modification	F	10	8000
H	Finalize manual	G	10	5000
I	Advertise	C,E	20	8000

(*) all dependencies are assumed to be FS-Finish to start And the following progress status:

ID	Task	Status	Actual Start(days)	Actual Duration(days)	Actual Costs (\$)
A	Meet with clients	100%			1500
B	Write SW	100%	+5 days	+10 days	9000
C	Debug SW	100%	+15 days	+5 days	2500
D	Prepare Draft manuals	100%	As per other delays		1000
E	Meet with clients	100%	As per other delays		1000
F	Test SW	100%	As per other delays		750
G	Make modification	0%	As per other delays		0
H	Finalise manual	0%	As per other delays		0
I	Advertise	10%	+5 on top of other delays		1000

Perform an analysis of the project status at week 13 ,using EVA. Use the CPI and SPI to determine project efficiency .Explain the process involved.

We organize the solution as follows:

1. Drawing the Gantt chart of the plan
2. Drawing the Gantt chart of the actual plan (progress status)

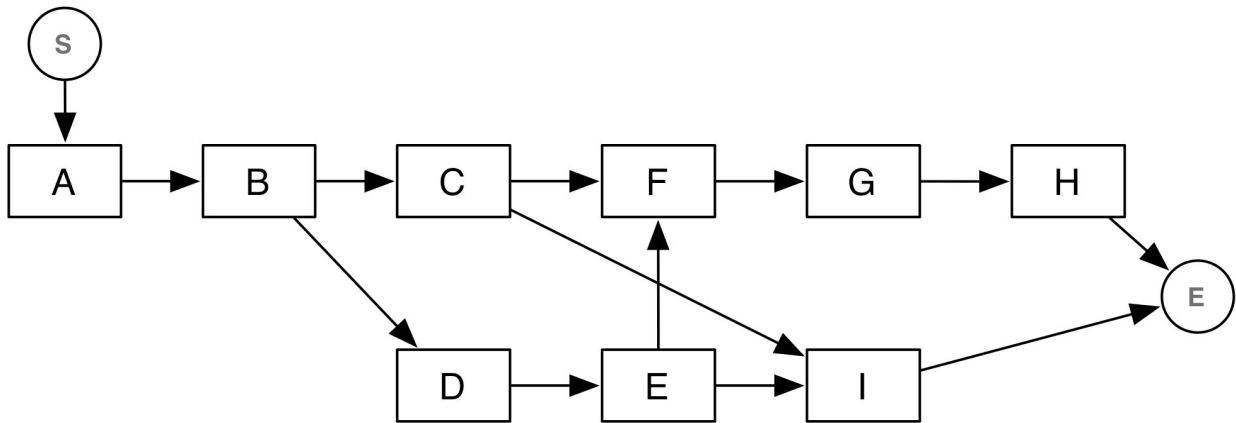
3. Perform the analysis (plot PV, AC, EV, CPI, SPI)

1. Drawing the Gantt chart of the plan

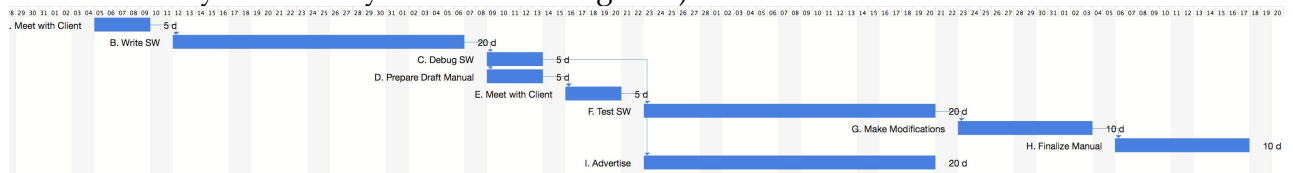
We start by drawing the network diagram using the information about immediate predecessors.

(This is not strictly necessary: the Gantt chart can be drawn directly, if you manage to take into account dependencies and durations at the same time, which should not be too complex.)

This is shown in the following figure, where we use the AON (Activity on Node) notation:

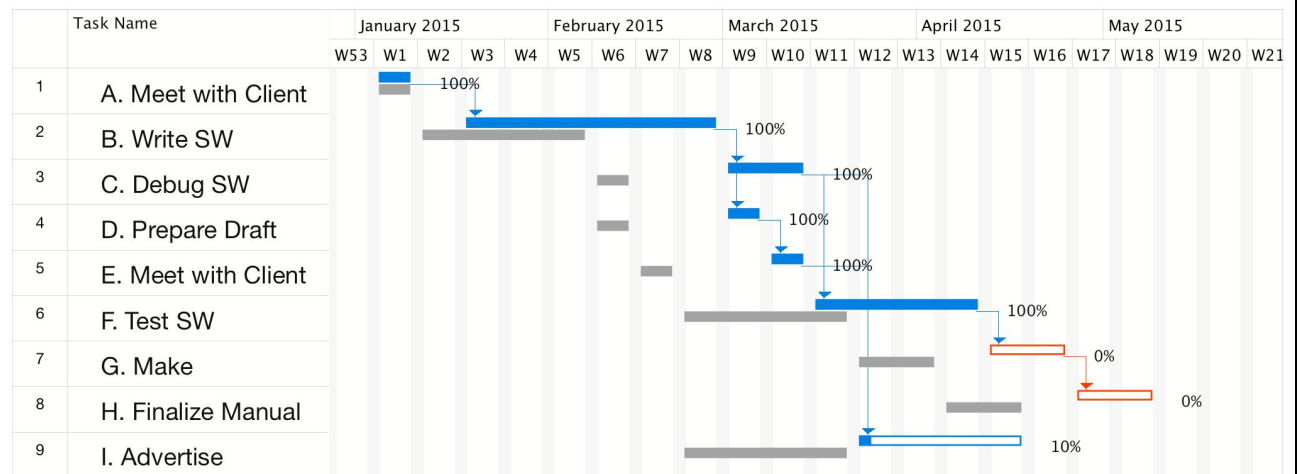


The Gantt chart can now be easily drawn, by taking into account the expected duration of each activity. The result is shown in the following diagram (notice that we are assuming the duration to be expressed in working-days and that we are using a “standard” calendar, in which Saturday and Sunday are non-working time):



2. Drawing the Gantt chart of the actual plan (progress status)

The actual Gantt chart can be drawn by taking into account the information about delays, variations in duration, and actual completion. The main point of attention (when doing this work manually), is taking into account the constraints. Gantt charting tools, fortunately, can do this for us automatically. The following figure shows the two plans, the baseline (or initial) plan, shown in the lower part of each activity and the actual plan, shown in the upper part of each activity:



As it can be seen, the delay on activity B delays all other activities in the plan. The activities marked in red are in the critical path.

3. Perform the analysis (plot PV, AC, EV, CPI, SPI)

To perform the assessment, we start by computing and plotting PV, AC, and EV. PV is the sum of planned costs. It is computed by determining for each reporting period, the cost associated to each activity and by summing and cumulating them over time. The following table summarizes the planned costs over time. It is computed as follows:

- Each column of the table represents one week (we show only the first 13 weeks)
- The planned costs of each activity is taken from the first table of the question
- For each activity, we compute the weekly cost (activity cost / duration in weeks) and accrue the cost for each week in which the activity lasts. For instance B has a total planned cost of 10000 and a duration of four weeks, from W2 to W5. Therefore we accrue 2500 in weeks W2 to W5 for B.
- We then compute the cumulative costs, by summing planned expenditure week by week.

	W1	W2	W3	W4	W5	W6	W7	W8	W9	W10	W11	W12	W13	Total
A Meet with client	500													500
B Write SW		2500	2500	2500	2500									10000
C Debug SW						1500								1500
D Prepare draft manual						1000								1000
E Meet with clients							1000							1000
F Test SW								500	500	500	500			2000
G Make modifications												4000	4000	8000
H Finalize manual														0
I Advertise								2000	2000	2000	2000			8000
Total	500	2500	2500	2500	2500	2500	1000	2500	2500	2500	2500	4000	4000	
Planned Value	500	3000	5500	8000	10500	13000	14000	16500	19000	21500	24000	28000	32000	

AC is the sum of the actual costs incurred into. It is computed by looking at the actual costs when they took place. Similar to the previous case:

- For each activity, we look at its actual costs (second table of the question) and split them evenly for the actual duration of the activity, up to the monitoring date (that is, the date in which the analysis is performed)

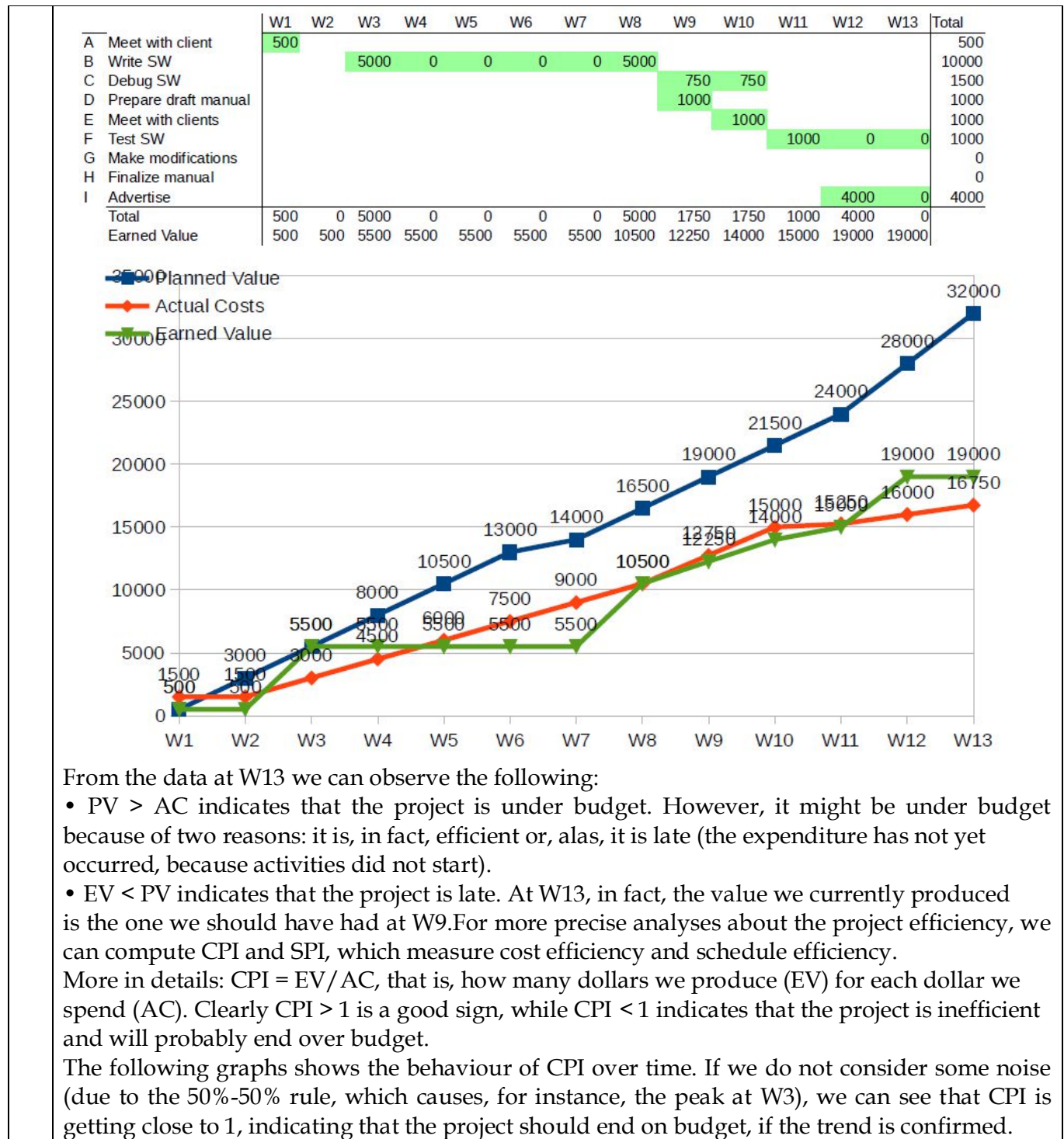
The result is shown in the following table:

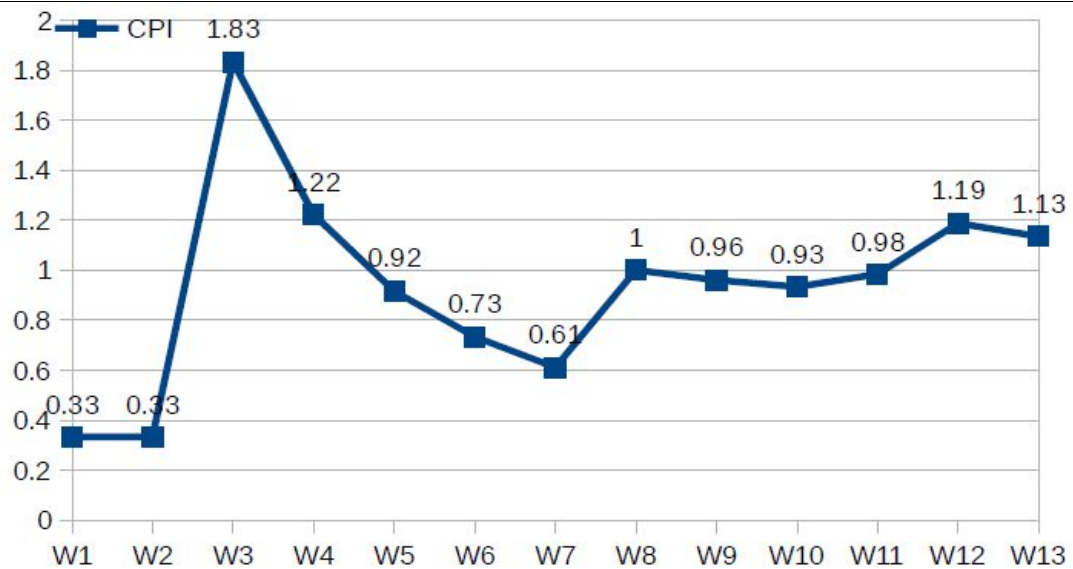
	W1	W2	W3	W4	W5	W6	W7	W8	W9	W10	W11	W12	W13	Total
A Meet with client	1500													1500
B Write SW		1500	1500	1500	1500	1500	1500							9000
C Debug SW								1250	1250					2500
D Prepare draft manual								1000						1000
E Meet with clients									1000					1000
F Test SW										250	250	250		750
G Make modifications														0
H Finalize manual														0
I Advertise											500	500		1000
Total	1500	0	1500	1500	1500	1500	1500	1500	2250	2250	250	750	750	
Planned Value	1500	1500	3000	4500	6000	7500	9000	10500	12750	15000	15250	16000	16750	

EV is the sum of the planned costs on the actual schedule.

There are different rules for computing EV. We use 50%-50% (50% of planned costs when an activity starts, the remaining 50%, when the activity ends).

The result is shown in the following table:





The SPI index measure the schedule: $SPI = EV/PV$ and indicates how much we produce (EV) with respect to what we thought we would produce. Also in this case $SPI > 1$ is a good sign (ahead of schedule), while $SPI < 1$ indicates that the project is late. In our example we should expect SPI to be < 1 , as it is, in fact, shown by the following diagram, which plots SPI over time: