SOFTWARE ENGINEERING

UNIT-I.

TWO MARKS

1.Define Software Engineering.

Software Engineering differs from traditional computer programming in that engineering-like techniques are used to specify, design, implement, validate and maintain software products within the time and budget constraints established for the project.

2.What are large projects?

- A large projects employees 5-20 programmers working for a period of 2-3 years.
- A system of 50000-10,000 lines of source code.

3.Define software quality.

Software quality refers to conformance to functional and performance requirements and development standards.

4.What are the factors that influence the software cost factors?

- Programmer Ability
- Product complexity
- Available Time
- Required Level of Reliability
- Level of Technology

5.Define Project planning.

- Project planning refers to the set of activities in project management.
- The major purpose of planning is to clarify goals, needs and constraints.
- Project planning includes defining the problem, determination of software scope, developing a solution strategy, planning the development process and estimation of software cost and resources.

6.What is hierarchical team?

- A project leader
- 5 to 7 Senior programmers
- 5 to 7 junior programmers

- The project leader assign tasks, attends reviews and walkthroughs, detects problem areas, balance work load, and participates in technical activities.

7.What is a software?

Software is nothing but a collection of computer programs, procedures, rules and associates documentation and data.

8.Define MBO.

Management By Objectives(MBO) is a technique by which employees set their own goals and objectives with the help of their supervisors, participate in the setting of their supervisor's goals, and are evaluated by meeting concrete, written objectives.

9.Define Waterfall Model.

- Waterfall Model is otherwise called as Phased Life Cycle
- Model/Classic Life Cycle Model/Linear Sequential Model.
- It was proposed by Winston Royce.
- It is the oldest and most widely used paradigm in software engineering.
- Waterfall model divides the software cycle into series of successive activities.
- Each phase requires well-defined input information, utilizes well-defined processes, and results in well-defined products.

10.What are the types of design?

- Internal design
- Architectural Design
- Detailed Design
- Data Design
- Procedural Design
- External Design
- User Interface Design.

FIVE MARKS

1.Discuss various size factors of a project.

Size factors

Various size factors of a project

S. no	Category	No. of programmers	Duration	Size	Example
1	Trivial	1	1-4 weeks	500 sources code	Games, puzzle solving
2	Small	1	1-6 months	1k-2k	Scientific applications
3	Medium	2-5	1-2 years	5k-50k	Inventory system
4	Large	5-20	2-3 years	50k-100k	IBM OS
5	Very large	100-1000	4-5 years	1M	Real time system
6	Extremely large	2000-10,000	5-10 years	1M-10M	Telecommunication

Trivial projects

- A trivial project involves 1 programmer working for 1-4 weeks, packed in a 10-20 subroutines.
- Programmers are often personal software.

Small Project

- A small project employs one programmer for 1 to 6 months
- Examples of small project include scientific applications written by engineers to solve numerical problems.

Medium-size project

- A medium-size software project employs 2-5 programmers working for two years.
- Project contains 10,000-50,000 statements packaged in 250-1000 subroutines.
- Example of medium size projects include Inventory system.

Large project

- A large projects employees 5-20 programmers working for a period of 2-3 years.
- A system of 50000-10,000 lines of source code.
- Example of large program includes large compliers.

Very large projects

- Very large projects employs 100-1000 programmers for the period of 4 years.

- Example of very large projects include large operating system.

Extremely large project

- An extremely large project employees 2000-5000 programmers for the period of upto 10 years.
- Example of extremely large projects include air traffic control.

2.Write a short note on qualities of software project.

Correctness

- A Correctness of a software satisfies its functional specification.

Reliability

- Reliability is a probability that software will operate as expected over a specified time interval.

Robustness

- Software is robust is it behaves reasonably even under unanticipated circumstance.
- Robustness and correctness are strongly related.

User-friendly

- User-friendly is measured in terms of ease of use response time correctness of result.

Verifiable

- A software is verifiable if its properties such as performance, correctness etc can be verified easily.

Maintainability

- It is an effort required to locate and fix errors in operating system.

Portability

- Software is portable if it can run in different hardware or software platforms

Modularity

- A system that is composed of modules is called modular system
- There are three advantages of using modular design
- Decomposition
- Compatibility
- Understanding

Testability

- Testability is an effort required to check module or a system

3.Describe various managerial issues in software development.

- Success of a software project is heavily depends on both managerial and technical activities equally
- Managers should control the resources and the environment.
- Other management responsibilities include developing business plans, recruiting and training employees.
- Management include, cost estimation techniques, resources allocation, budgetary control, assessing progress, establishing quality assurance procedures, communicating with customers, developing contractual agreements with customers etc.
- Some of the problems identified in management
- Planning for software engineering projects is generally poor.
- Procedures and techniques are poor
- The accountability is poor.
- Decision rules are not available.
- Some of the Methods for solving the problems
- Educate and train developers.
- Enforce the use of standards.
- Define objectives in terms of deliverables
- Define quality in terms of deliverables.
- Establish success priority criteria.

TEN MARKS

1.Explain about the factors that influence quality and productivity.

Some factors that influence quality and productivity are discussed below.

- Individual Ability
- Team Communication
- Product complexity
- Application programs
- Appropriate notations
- Level of reliability
- Available time

- Required skills.
- Adequacy of training
- Management skills
- Systematic approaches
- Change control
- Level of technology
- Appropriate goals

Individual Ability

- Productivity and quality are direct functions of individual ability and effort.
- Familiarity of the individual with the particular application area.

Team Communication

- Proper communication among the programmers should be established in order to complete the software product successfully on time within the budget and with the expected quality.

Product complexity

There are three levels of product complexity they are

1.Application programs

2.Utility programs

3.System programs

Application programs

- Application program include usually database programs written in high-level programming languages.

Utility programs

- Utility programs include compliers, assemblers and loaders.

System programs

- System programs include data communication packages and operating system routines.

Appropriate notations

- Programming languages should provide good notation for implementation phase of the development of the software products

Level of reliability

- Software reliability is defined as the probability of failure free operations of a program for a specified time in specified environment

Available time

- Time is an important factor in developing a software product with expected quality.

Required skills.

- The software engineer should posses a good communication skill , talk and diplomacy as well as knowledge in the application area.
- Implementation of software require writing program with no errors in syntax

Adequacy of training

- Sufficient training should be given to the team members, who are going to involve in the development of software product.

Management skills

- The management skills are extremely needed for the successful development of a software product.

Systematic approaches

- It is quite unreasonable to expect that a single approach to soft development and maintenance will ever be adequate to cover all situations.
- At this point in the evolution of software engineering, it is often not clear which of the various approaches to software development should be used in which situations

Change control

- Some projects experience constantly changing requirements, which can quickly undermine project morale.
- Notations and procedures that provide the ability to trace and assess the impact of proposed changes are necessary to make visible the true cost of apparently small changes to source code.
- Usage of appropriate notations and techniques makes controlled changes possible without degrading the quality of work products

Level of technology

- Modern programming languages provide improved facilities for data definition and data usage, improved constructs for specifying control flow, better modularization facilities, user-defined exception handling and facilities for concurrent programming.

Appropriate goals

- The primary goal of software engineering is the development of software products; that are appropriate for their intended use. Every software product should provide optimal levels of generality, efficiency and reliability.
- An appropriate trade-off between productivity and quality factors can be achieved by adhering to the goals and requirements established for the software product during project planning.

2.Explain in detail about planning the development process.

Life-cycle models include, the phases model the cost model the prototype model and the successive versions models.

The phased life-cycle model

The phased life cycle model consists of the following phases they are,

1.Analysis phase

2.Design phase

3.Implementation phase

4.System testing

5Maintenance phase

Analysis phase	Design phase	Implementation phase	System test	Maintenance
Planning requirements definition Verify	 Architectural design verify	 Code debug Unit test verify	 Integration acceptance	 Enhancement Adapt and fix

Analysis phase

- Analysis phase consists of two sub phases planning and requirements definition

Design phase

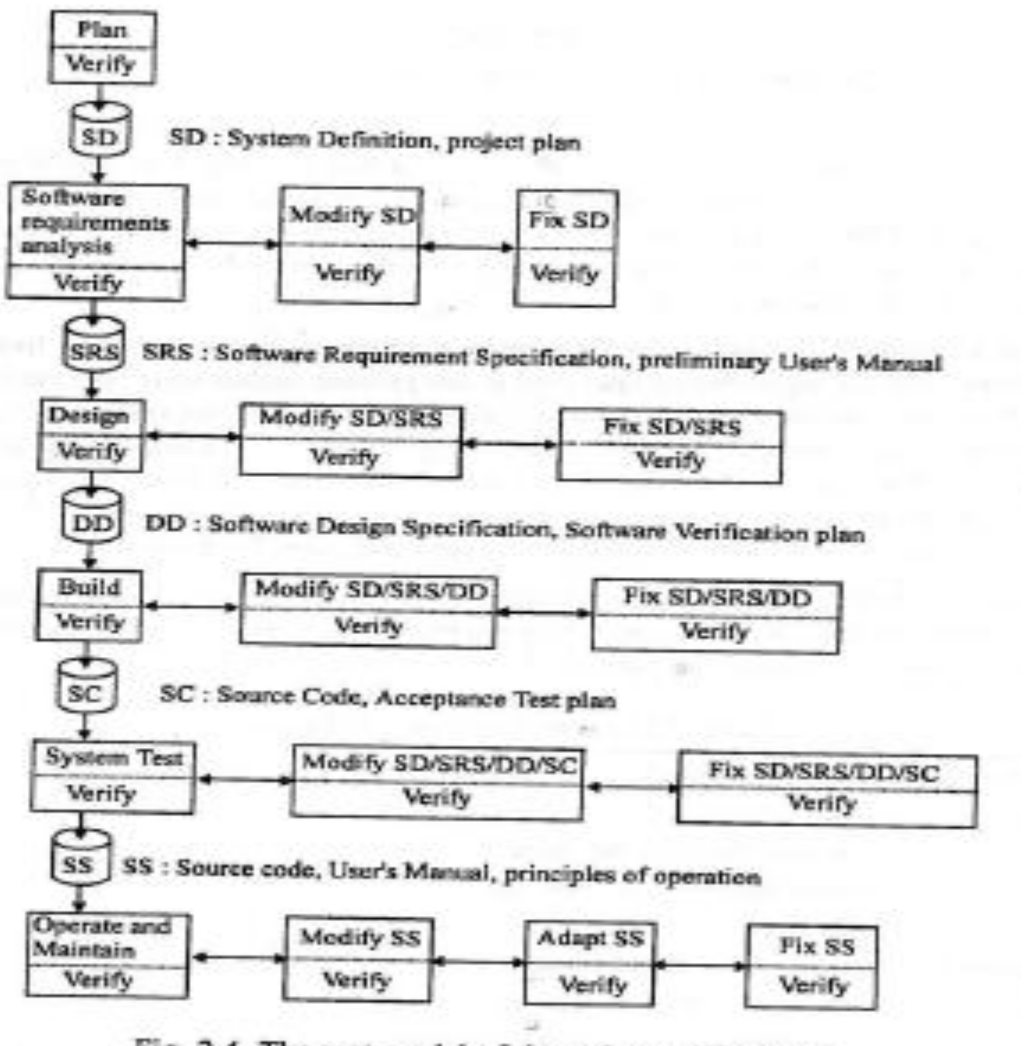
- The design phase consists of architectural design and detailed design

Implementation phase

- This phase involves source code, debugging, documentation and unit testing of the source code.
- System testing In involves integration testing and acceptance testing

Maintenance phase

- Maintenance activities include enhancement of capacities.


Plan
Verify
SD
SD : System Definition, project plan
Software requirements analysis
Verify
Modify SD
Verify
Fix SD
Verify
SRS
SRS : Software Requirement Specification, preliminary User's Manual
Design
Verify
Modify SD/SRS
Verify
Fix SD/SRS
Verify
DD
DD : Software Design Specification, Software Verification plan
Build
Verify
Modify SD/SRS/DD
Verify
Fix SD/SRS/DD
Verify
SC
SC : Source Code, Acceptance Test plan
System Test
Verify
Modify SD/SRS/DD/SC
Verify
Fix SD/SRS/DD/SC
Verify
SS
SS : Source code, User's Manual, principles of operation
Operate and Maintain
Verify
Modify SS
Verify
Adapt SS
Verify
Fix SS
Verify
Fig. 2.4. The cost model of the software life cycle

The Cost Model

- Another view of the software life cycle can be obtained by considering the cost performing the various in a software project.
- The prototype life-cycle model

- A prototype is a mock-up or model of a software product.
- Prototyping steps

.**The Successive version model**

- Software project development by the method of successive version an extension of prototyping model.
- In this approach each successive version of the product is a functioning system capable of performing useful work.

Unit-II

Two marks

1.What are the Software Cost Estimation Techniques?

There are four widely used techniques available for software cost estimation. They are

1.Expert Judgment

2.Delphi Cost Estimation

3. Work Breakdown Structure

4.CoCoMo or Cost Constructive Model or Algorithmic Cost Models.

2.What are the factors that influence the software cost factors?

- Programmer ability
- Product size
- Available time
- Required reliability
- Level of technology.

3. Write the format of a requirement specification document.

Section 1: Product overview and summary

Section 2: Development, Operating and Maintenance Environments

Section 3: External Interfaces and data flow

Section 4: Functional Requirements

Section 5: Performance Requirements

Section 6: Exception Handling

Section 7: Early subsets and Implementation Priorities

Section 8: Foreseeable Modifications and Enhancements

Section 9: Acceptance criteria

Section 10: Design Hints and Guidelines

Section 11: Cross-Reference Index

Section 12: Glossary of terms

4.Define SRS

- A software requirement specification is a set of documents that contains the concise and clear specification of all the requirements of the proposed system

5.What are the activities include in software maintenance?

- Software maintenance typically requires 40 to 60 percent and in some cases as much as 90 percent of the total life - cycle effort devoted to software product.
- Maintenance activities include
- 1. Enhancements to the product.
- 2. Adapting the product to new processing environments.
- 3. Correcting problems.

6.Define Work Break Down Structure.

- A work break down structure is a estimation tool
- A most widely used estimation
- A WBS is a hierarchical chart that accounts for the individual parts of a system

7.What is Detailed CoCoMo or Complete CoCoMo?

- In complete CoCoMo large system is divided into several sub system with different characteristic.
- This cost of each sub system is estimated and finally the sum of the estimates give the total cost of the project.

8.What are the need for SRS?

- A basis purpose of developing SRS is to bridge the communication between the client or users and developers.
- The software should be developed to satisfy all the needs of users under the constrains specified by the client.

9.Define PSA.

The Problem Statement Analyzer (PSA)

- The Problem Statement Analyzer (PSA) is an automated analyzer for processing requirements stated in PSA. PSL description the PSA system can provide reports in four categories.
- Data-base modification reports
- References reports

- Summary reports
- Analysis reports

10. Define the Problem Statement Analyzer (PSA) is the PSL processor.

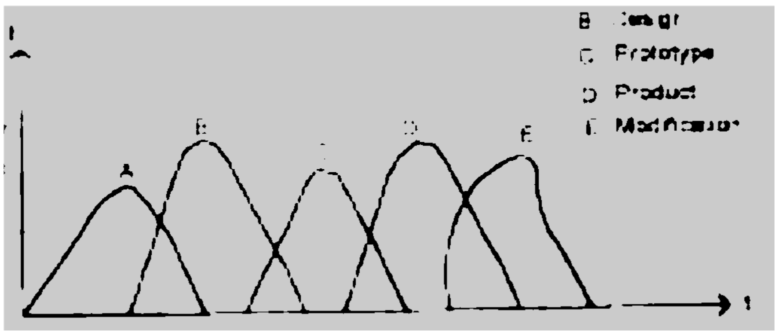
- The Problem Statement Analyzer (PSA) is the PSL processor
- PSL is based on a general model of systems.
- This model describes the system as a set of objects, where each object may have properties and each property may have property values. Objects may be inter-connected; the connections are called relationships.

Five marks

1.Discuss about the staffing level estimation.

Staffing-Level Estimation

- The number of personnel required throughout a software development project is constant.
- The number of personnel required during each phase of the project differs from one project to the other.
- Typically, planning and analysis are performed by a small group of people, architectural design by a larger group and detailed design by a still larger number of people.
- The early phase of maintenance may require numerous personnel, but the number should decrease in a short time.
- If there is no major enhancement or adaptation, then the number of personnel for maintenance should remain small.
- The Rayleigh curve is specified by two parameters, td. the time at which the curve reaches its maximum value, and k, the total area under the curve, which represents the total number of full-time equivalent personnel required at that instant of time.



- According to Putnam, the time at which the Rayleigh's curve reaches its maximum value.
- corresponds to the time system testing and product release for many software products.
- The area under the Rayleigh curve in any interval represents the total effort expended in that interval.
- Boehm observes that the Rayleigh curve is a reasonably accurate estimator of personnel requirements for the development cycle from architectural design through implementation and system testing if the portion of the curve between 0.3 td and 1.7 td.
- PM - estimated number of programmer months for product development (excluding planning and analysis)
- TDEV - estimated development time
- Given these two factors, the number of full-time software personnel, FSP, required at any particular time t, where t is in the range 0.3 td to 1.7td can be computed.

- Boehm also presents tables that specify the distribution of effort and schedule in a software development project.
- The total number of programmer-months and total development time can be used to obtain an estimate of the actual number of programmer months and elapsed time for each activity
- An estimate of the number of full-time development can be obtained by dividing the numbers of programmer - months required by the elapsed time available.

2.Write about regular expressions in formal specification techniques

Regular expressions

- Regular expressions can be used to specify the syntactic structure of symbol strings.
- As many software products involve processing of symbol strings, regular expressions provide a powerful and widely used notation in software engineering.

- Every set of symbol strings specified by a regular expressions defines a formal language. Regular expressions can thus be viewed as language generators.

The rules for forming regular expressions are as follows.

.

1. Axioms: The basis symbols in the alphabet of interest form regular expressions.
2. Alternation: If Rl and R2 are regular expressions, then (R1/R2) is a regular expression.
3. Composition: If Rl and R2 are regular expressions, then (R1.R2) is a regular expressions.
4. Closure: If Rl is a regular expressions, then (Rl)* is a regular expression.
5. Completeness: Nothing else is a regular expression.

3.Write about the relational notations in formal specification technique regular.

Relational Notations

- Implicit equations: Implicit equations specify the properties of a solution without stating a solution method.
- Matrix inversion is specified as follows

$$Mx_{M}' = I + E \ (31)$$

- Matrix inversion has the property that the original matrix (M) multiplied by its inverse (M^{1}) yield an identify matrix.
- This property of the matrix inverse is specified in equation .
- I denotes the identify matrix and E specifies allowable computational errors Complete specification of matrix inversion must include items such as matrix size, type of data elements and degree of sparseness (ie) how many elements in the matrix are zeroes Given a complete functional specification for matrix inversion, design involves specifying a data structure and an algorithm for computing the inverse.

  Implicit equations
- Specification of a square root function, SQRT, can be stated as

$$(0<=_{x}<=y)\ [ABS(SQRT(X)**2-X)<E]\ (3.2)$$

- Not all implicitly specified problems are guaranteed to have algorithmic solution

  Recurrence relations

- A recurrence relation consists of an initial part called the basic and one or more recursive parts.
- The recursive parts describe the desired value of a function in terms of other values of the function for example, successive Fibonacci numbers are formed as the sum of the previous two Fibonacci numbers.
- The initial part specifies the first two terms in the Fibonacci series and they are 0 and 1.

  F(0) = 0

  F(l)=l

  F(N) = F(N-1) + F(N-2) for ail N>1
- Recurrence relations are easily transformed into recursive programs."That doesn't mean every recursive specification should;d be implemented as a recursive programs.

Algebraic Axioms

- Mathematical systems are defined by axioms. The axioms specify fundamental properties of a system and provide a basis for deriving additional system and provide a basis for deriving additional properties that are implied by the axioms.
- These additional properties are called theorems. The set of axioms must be complete and consistent, i.e. it must not be possible to prove contradictory results.
- The axiomatic approach can be used to specify functional properties of software systems.
- The intention is to specify the fundamental nature of the system by stating a few basic properties.
- This approach can be used to specify abstract data types. A data type is characterized as a set of objects and a set of permissible operations on those objects.
- The term "abstract data type" refers to the fact that permissible operation on the data objects are emphasized while representation details of the data objects are suppressed.

Ten marks

1.Explain the various cost estimation techniques.

There are four widely used techniques available for software cost estimation. They are

1.Expert Judgment

2.Delphi Cost Estimation

3. Work Breakdown Structure

4.CoCoMo or Cost Constructive Model or Algorithmic Cost Models.

Expert Judgment

- The most widely used cost estimation technique is expert judgment.
- It is a top-down estimation technique
- This method involves consulting one or more experts.
- The experts provide estimates using their own methods and experience
- Expert judgment relies on the experience background and business sense
- The expert must be confident that the project is similar to previous one.

Delphi Cost estimation

- This methods involves more interaction and communication between participants.
- The Coordinator presents a specification form to each experts
- Coordinator calls a group meeting in which experts discuss estimation
- Each expert fills out forms.
- 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
- Process repeated until coordinator is satisfied.

Work Break Down Structure

- A work break down structure is a estimation tool
- A most widely used estimation
- A WBS is a hierarchical chart that accounts for the individual parts of a system

CoCoMo or Cost Constructive Model or Algorithmic Cost Models

- It was developed by Barry Boehm in 1987, by analyzing more than 100 projects.
- It has 3 forms
- Basic CoCoMo models
- This divides project complexity into three.

They are

1.Organic

2.Semi detached

3.Embedded

Effort= α *(KDSI)

M = 2.8*(KDSI)** 1.20 = 2.8* (10)** 1.20 = 44.4

TDEV = 2.5 * (PM) * * 0.32 ' = 2.5 * (44) * * 0.32 = 8.4

KDSI – Thousands of LOC(Lines Of Code)

	α	β	γ	δ
Organic	2.4	1.05	2.5	0.38
Semi-Detached	3.0	1.12	2.5	0.35
Embedded	3.6	1.20	2.5	0.32

Intermediate CoCoMo

- In intermediate CoCoMo the Cost estimation for the software development is calculated by considering 15 other factors.

Detailed CoCoMo or Complete CoCoMo

- In complete CoCoMo large system is divided into several sub system with different characteristic.

- This cost of each sub system is estimated and finally the sum of the estimates give the total cost of the project.

2.Explain briefly about Software Requirement Specification[SRS]

Software Requirement Specification[SRS]

- A software requirement specification is a set of documents that contains the concise and clear specification of all the requirements of the proposed system
- The format of a requirement specification document is

Section 1: Product overview and summary
Section 2: Development, Operating and Maintenance Environments
Section 3: External Interfaces and data flow
Section 4: Functional Requirements
Section 5: Performance Requirements
Section 6: Exception Handling
Section 7: Early subsets and Implementation Priorities
Section 8: Foreseeable Modifications and Enhancements
Section 9: Acceptance criteria
Section 10: Design Hints and Guidelines
Section 11: Cross-Reference Index
Section 12: Glossary of terms

Need for SRS

- A basis purpose of developing SRS is to bridge the communication between the client or users and developers.
- The software should be developed to satisfy all the needs of users under the constrains specified by the client.
- Most often the clients does not understand the software and the software development process and the developers does not understand the clients applications area and problems in such case SRS act as a solution

- External Interfaces such as user displays and report formats, a summary of user commands and report options, data flow diagrams and data dictionary. High level data flow diagrams and a data dictionary are derived.
- Data flow diagrams specify data sources and data sinks, data stores, transformations to be performed on the data, and the flow of data between sources, sinks, transformations and stores. A data store is a conceptual data structure in the sense that physical implementation details are suppressed; only the logical characteristics of data are emphasized in a data flow diagrams.
- The arcs in a data flow diagram specify data flow. They are labeled with the names of data items. The characteristics of these data items are specified in the data dictionary.
- Data flow diagrams are not concerned with decision structure or algorithmic details. Data flow diagrams can be used at any level of detail. They can be hierarchically decomposed. The inner workings of the functional nodes are specified using additional data flow diagrams.

- The actions to be taken and the messages to be displayed in response to undesired situations or events are specified in the software Requirements specification.

- A table of exception conditions and exception responses should be prepared. Categories of possible exceptions include temporary resource failure.
- The software Requirements specification specifies early subsets and implementation priorities for the system under development. Software products are sometimes developed as a series of successive versions.
- Each successive version can *t* incorporate the capabilities of previous versions and provide additional processing functions. Sometimes the customer may desire to have early delivery of the software product with limited capability and may wish the successive versions to provide increasing levels of capabilities.

- The capabilities that must be included to each version must be planned initially. The subsets of the capabilities that are to be included to each successive version is specified in this section.
- Depending on the importance of the system capabilities may be

3.Disscuss briefly about the languages and processors for requirements specification.

A number of special-purpose languages and processors have been developed to permit concise statement and automated analysis of requirements specifications for software. Some specification languages are graphical in nature.

PSL/PSA

- The Problem Statement Language (PSL) was developed by Professor Daniel Teichrow at the University of Michigan,
- The Problem Statement Analyzer (PSA) is the PSL processor. PSL is based on a general model of systems.
- This model describes the system as a set of objects, where each object may have properties and each property may have property values. Objects may be inter-connected; the connections are called relationships.
- The general model is specialized to information systems by allowing only a limited number of predefined objects, properties and relationships.
- In PSL, system descriptions can be divided into eight major aspects:
- System input/output flow.
- System structure
- Data structure
- Date derivation
- System size and volume
- System dynamics
- System properties

The Problem Statement Analyzer (PSA)

- The Problem Statement Analyzer (PSA) is an automated analyzer for processing requirements stated in PSA. PSL description the PSA system can provide reports in four categories.
- Data-base modification reports
- References reports
- Summary reports
- Analysis reports
- Data-base modification reports list changes that have been made since the last report together with diagnostic and warming messages, these reports provide a record of changes for error correction and recovery. Reference reports include.
  - Name list report
  - Formatted problem statement Report and
  - Dictionary Report
- Name List Report lists all the objects in the database with types and dates of last change. The Formatted problem Statement Report shows properties and relationships for particular object.

UNIT=III
TWO MARKS

1. Define Modularity

- There are many definitions of the term "module". The modular systems incorporate collections of abstractions in which each functional abstraction, each data abstraction and each control abstraction handles a local aspect of the problem being solved.
- Modularity enhances design clarity, which in turn eases, implementation, debugging, testing, documenting and maintenance of the software product.

2. What are the properties of modular system?

Desirable properties of a modular system include.

1. Each processing abstraction is a well-defined subsystem that is potentially useful in other applications.
2. Each function in each abstraction has a single, well defined purpose.

3.Defne Abstraction

Abstraction is the intellectual tool that allows us to deal with concepts apart from particular instances of these concepts. During requirements definition and design, abstraction permits separation of the conceptual aspects of a system from the implementation details. We can, for example, specify the FIFO property of queue or the LIFO property of a stack without concern for the representation scheme to be used in implementing the stack or queue.

4. Define Information Hiding .

Information hiding is a fundamental design concept for software, When a software systems is designed using the information hiding approach, each module in the system hides the internal details of its processing activities and module communicate only through well-defined interfaces.

5. Define Verification

Verification is a fundamental concept on software design. Design is the bridge between customer requirements and an implementation that satisfies those requirements.

A design is verifiable if it can be demonstrated that the design_ will result in an implementation that satisfies the customer's requirements this is typically done in two steps.

(1) verification that the software requirements definition satisfies the customer's needs.

(2) verification the design satisfies the requirements definition.

6. What are the characteristics of a software module

Modules and modularization criteria

Architectural design has the goal of producing well structured modular software systems. A software module to be a named entity having the following characteristics.

1. modules contain instructions, processing logic and *daik* structures
2. modules can be separately, compiled and stored in a library
3. modules can be included in a program.
4. module segments can be used by invoking a name and some parameter.
5. modules can use other modules.

7.What is walkthrough?

A structured walkthrough is an indepth technical revew of some aspect of a software system. Walkthrough can be used at any time, during any phase of a software project. Thus, all or any part of the software requirements, the architechtural design specifications, the detailed design specification can be reviewed at any stage of evolution.

8.What is HIPO diagrams?

HIPO diagrams (Hierarchy-Process-Input-Output) were developed at IBM as design representation schemes for top-down software development and as external documentation aids for released products.

Five marks

1.Distinguish between Black box testing and white box testing.

Black box

- Black box requires no knowledge about the minute internal working of the module concerned.
- This test is conducted to check the total functionality of the module.
- If the module is not producing results as expected the error has to be identified and rectified.
- This type of testing is not done normally for modules.
- It is applied on the product as a whole,

White box

- White box testing requires complete knowledge of the individual modules, all its independent paths, all logical and decision making checks and also its test case design.
- This type of test is an exhaustive test.
- Such exhaustive test are necessary because of the varied nature of software errors, logical errors, path errors, typo graphical errors, syntax errors and validation errors.

2.Explain about coupling and Cohesion

- A fundamental goal of software design is to structure the software product so that the number and complexity of inter connection between modules is minimized.
- The strength of coupling between two modules is influenced by the complexity of the interface, the type of connection and the type of communication
- Modification of a common data block or control block may require modifications of all routines that are coupled to that block. If modules communicate only by parameters and if the interface between modules remain fixed, the internal details of modules can be modified without having to modify the routine that use the modified module.
- Connection established by referring to other module names are more loosely coupled than connections established by referring internal elements of other modules.
- The degree of coupling is lowest for data communication, higher for control communication and highest for modules that modify other modules.
- Coupling between modules can be ranked on a scale of strongest to weakest as follows.
- content coupling
- common coupling
- control coupling
- stamp coupling
- data coupling

- Content coupling occur when one module modifies local data values or instructions in another module. Content coupling can occur is assembly language programs.
- Common coupling module are bound together by global data structures. For instance, common coupling results when all routines in a FORTRAN program reference a single common data block. <,
- Control coupling involves passing control flags (as parameters or globals) between modules so that one module controls the sequence of processing steps in another module.
- Stamp coupling is similar to common coupling, except that global data items are shared selectively among routines that require the data**.**

Data coupling involves the use of parameter lists to pass data items between routines. The most desirable form of coupling between modules is a combination of stamp and data coupling.

3.Explain about Cohesion

Cohesion

- The internal cohesion of a module is measured in terms of the strength of binding of element with in the module
- . Cohesion of element occurs on the scale of weakest to strongest in the following order

1. coincidental cohesion**s**
2. logical cohesion
3. temporal cohesion
4. communication cohesion
5. sequential cohesion
6. functional cohesion

7. information cohesion.

Coincidental cohesion occurs when the elements within a module have no apparent relationship to one another. This results when a large, monolithic program is "modularized" by arbitrarily segmenting the program into several small modules.

Logical cohesion implies some relationship among the elements of the module, as for example, in a module that performs all input and output operations or in a module that edits all data.

Module with **temporal cohesion** exhibit many of the same advantages as logically bound modules.. A typical example of temporal cohesion is module that performs program initialization.

The elements of a module possessing **communicational cohesion** refer to the same set of input and or output data for example, "print and punch the output file" is communicational bound. Communicational binding is higher on the binding scale than temporal binding because the elements are executed at one time and also refer to the same data.

Sequential cohesion of elements occurs when the output of one element is the input for the next element. For example, "Read Next transaction and Update Master File" is sequentially bound. Sequential cohesion is high on the binding scale because the module structure usually bears a close resemblance on to the problem structure.

Functional cohesion is a strong and hence desirable type of binding of elements in a module because all elements are related to the performance of a single function. Example are "Computer Square Root"; "Obtain Random Number" and "Write Record to Output File".

Information cohesion of elements in a module occurs when module contains a complex data structure and several routines to manipulate the data structure.

4.Discuss about Real-Time and Distributed System Design

- Many of the popular design "methodologies" were developed-as design techniques for applications programs, operating systems and utility programs.
- These methods support concepts such as hierarchical decomposition, modularity, information hiding and data abstractions.
- The design of distributed systems is further complicated by the need to allocate network functionality between hardware and software components of the network.
- By definition, real-time systems must provide specified amounts of computation with in fixed time intervals.

Real-time systems typically sense and control external devices, respond to external events, and share processing time between multiple tasks processing demands are both cyclic and event-driven in nature.

- Event-driven activities may occur in bursts, thus requiring a high ratio of peak to average processing.
- Real-time systems often form distributed networks, local processors may be associated with sensing devices and actuators.
- A real-time network for process control may consist of several minicomputers and micro computers connected to one or more large processors.
- Each small processor may be connected to a cluster of real-time devices.

- Process control systems often utilize communication networks having fixed, static. topology and known capacity requirements
- . Higher-level issues of networking, performance and reliability must be analyzed and designed before the component nodes or processes are developed.

TEN MARKS

1.Explain in detail about design techniques.

- The design process involves developing a conceptual view of the system, establishing system structure identifying data streams and data stores, decomposing high level functions into sub functions, establishing relationships and interconnection among components.
- Developing a conceptual view of a software system involves determining the type of system to be built.
- **Backtrackin**g is fundamental to top-down design. In order to minimize backtracking, many designers advocate a mixed strategy that is predominately top-down but involves specifying the lowest-level modules first.

Stepwise Refinement

- Stepwise refinement is a top-down technique for decomposing a system from high-level specifications into more elementary levels. Stepwise refinement is also known as “step wise program development" and "successive refinement". It involves the following activities.

1) Decomposing design decisions to elementary levels.

2) Isolating design aspects that are not truly interdependent.

3) Postponing decisions concerning representation details as long as possible.

4) Carefully demonstrating that each successive step in the refinement process is a faithful expansion of previous steps.

Levels of abstraction

- Levels of abstraction was originally described by Dijkstra as a bottom-up design technique in which an operating system was designed as a layering of hierarchical levels starting at level and building up to the level ol processing independent user programs.
- Internal functions are hidden from high levels: they can only be invoked by functions on the same level. The internal functions arc used to perform tasks common to the work being performed on that level of abstraction.
- Each level of abstraction performs a set of services for the functions on the next higher level of abstraction.
- Each level of abstraction has exclusive use of certain resources (I/O devices, data structures) that other levels are not permitted to access.

Structured design

- Structured design is a top-down technique for architectural design of software systems.
- The basic approach is structured design is systematic conversion of data-flow diagrams into structure charts.
- A well designed systems exhibits a low degree of coupling between modules and a high degree of cohesion among elements in each module.

- **The first step** in structured design is review and refinement of the data flow diagram(s) developed during requirements definition and external design.
- **The second step** is to determine whether the system is transform-centered or transaction driven and to derive a high level structure chart based on this determination
- **The third step** in structured design is decomposition of each subsystem using guidelines such as coupling, cohesion, information hiding, levels of abstraction, data abstraction and the other decomposition criteria
- Decomposition of processing functions into modules should be continued until each module contains no subset of element that can be used alone and until each module is small enough that its entire implementation can be grasped at once.
- **Data dictionary** can be used with a structure chart to specify data attributes relationships among data items and data sharing among modules in the system.
- The "Scope of control" of a module is that module plus all modules that are subordinate to it in the structured chart.
- **Integrated Top-down development**
- Integrated top-down development integrates design, implementation and testing.
- Using integrated top-down development, design proceeds top-down from the highest-level routines; they have the primary function of co-ordinations and sequencing the lower-level routines

Jackson structured programming

Jackson Structured Programming was developed by Michael Jackson as a systematic technique for mapping the structure of a problem into a program structure to solve the problem. The mapping is accomplished in three steps.

1. The problem is modeled by specifying the input and output data structures using tree structures diagrams.
2. The input-output model is converted into a structural model that contains the operations needed to solve the problem.

Input and output structures are specified using a graphical notation to specify data hierarchy, sequence of data.

1. OPEN FILES
2. CLOSE FILES
3. STOP RUN
4. READ A RECORD INTO PART-NUM, MOVMNT
5. WRITE HEADING
6. WRITE NET-MOVEMENT LINE
7. SET NET-MOVMNT TO ZERO
8. ADD MOVMNT FROM NET-MOVMNT
9. SUBTRACT MOVMNT FROM NET-MOVMNT

2.Explain briefly about Design notations.

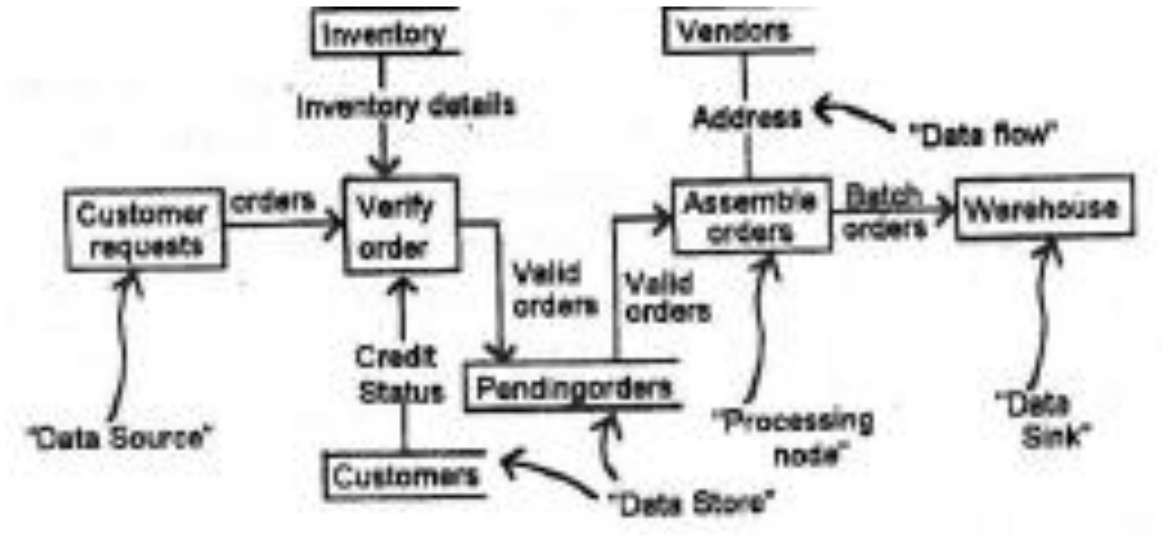
- Good notation can clarify the interrelation ship and interactions of interest, while poor notation can complicate and interface with good design practice.
- At least three levels of design specification exist, external design specification, which describe the external characteristics of a software system; architechtural design specifications; which describe the structure of the system; and detailed design specifications, which

describe control flow, data representation and other algorithmic details within the module.

- Notations used to specify the external characteristics, architectural structure and processing details of a software system include data flow diagrams, structure chart. HIPO diagram procedure specifications pseudo code. Structural English, and structural flowcharts.

Data flow diagrams

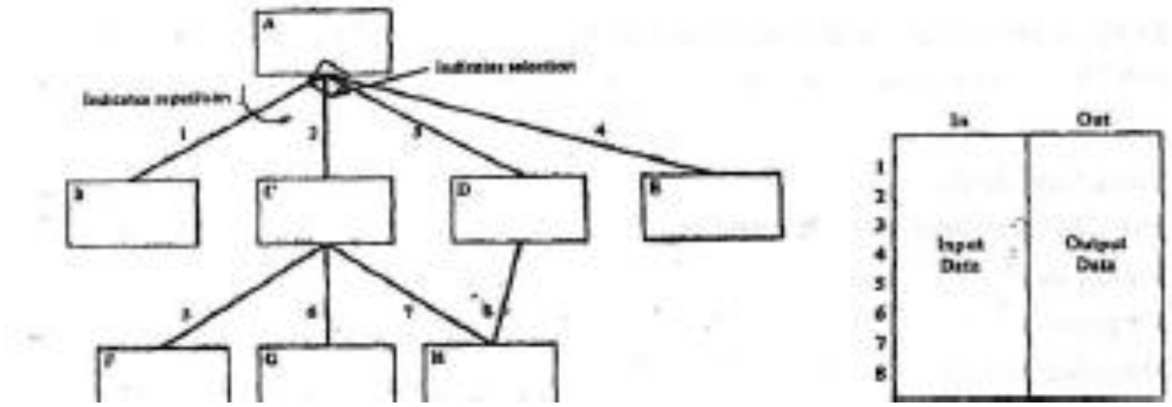
- Data flow diagrams are excellent mechanisms for communicating with customers during requirement analysis they are also widely used for representation of external and top-level internal design specification.
- A data flow diagram might represent data flow between individual statements or blocks of statements in a routine, data flow between concurrent processes or data flow in a distributed computing system.



Structure charts

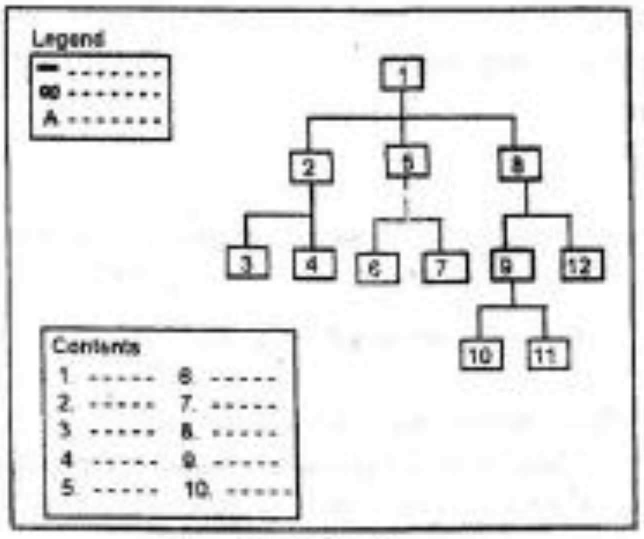
- Structure charts are used during architectural design to document hierarchical structure, parameters and interconnection in a system.

- A structure chart from a flow chart in two ways: a structure chart has no decision boxes and the sequential ordering of tasks inherent in a flow chart.



HIPO Diagrams

- HIPO diagrams (Hierarchy-Process-Input-Output) were developed at IBM as design representation schemes for top-down software development and as external documentation aids for released products.



Procedure template

As design progresses, the information on levels 2,3 and 4 can be included in successive steps.

Procedure Name

Part of (subsystem name & number)

Called by: LEVEL 1

Purpose:

Designer/Date(s)

Parameters: (names, modes, attributes, purposes)

Input Assertion: (Preconditions) LEVEL 2

Output Assertion: (Post conditions)

Global: (names, modes, attributes, purposes shared with)

Side effects:

Local Data Structures: (names, attributes,

purposes) Exceptions: (conditions, responses)

LEVEL 3

Timing

Constrain

ts: Other

Limitatio

ns

Procedure Body: (pseudo code, Structured English, Structured flow chart, decision table) LEVEL 4

Format of procedure template.

Pseudo code

Pseudo code notation can be used in both the architectural and detailed level of abstraction using pseudo code, the designer describes system characteristics using short. concise, English language phrases that are structural by key words such as If-then-Else, While-Do, and End.

Using the top-down design strategy, each English phrase is expanded into more detailed pseudo code until the design specification reaches the level of detail of the implementation language.

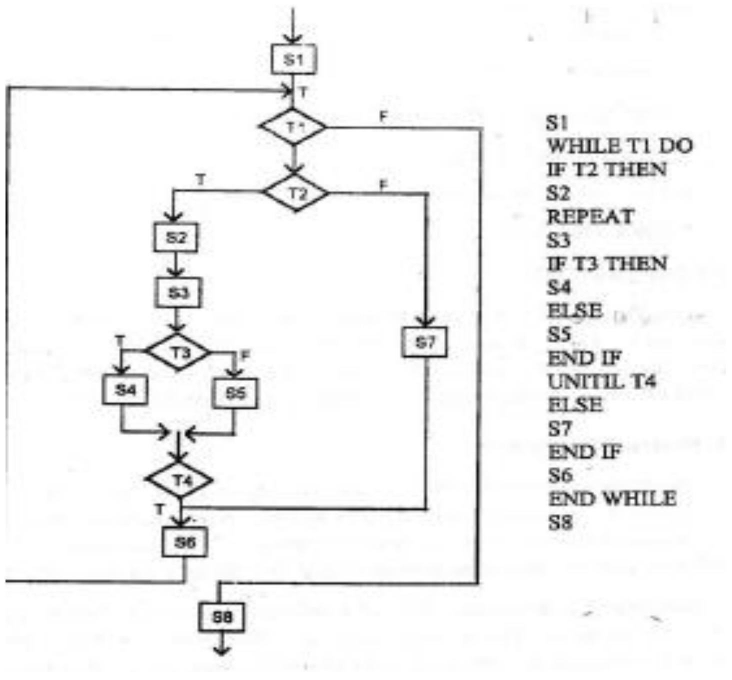
Pseudo code can replace flowcharts and reduce the amount of external documentation required to describe a system

```
    INITIALIZE tables and counters, OPEN files
    READ the first text record
    WHILE there are more text records DO
    WHILE there are more words in the text record DO
    EXTRACT the next record
    SEARCH word table for the extracted word
    IF the extracted word is found THEN
    INCREMENT the extracted word's occurrence count
    ELSE .
    INSERT the extracted word into word table
    ENDIF
    INCREMENT the words processed counter
END WHILE at the end of the text record
END WHILE when all text records have been processed
PRINT the word_table and the words_processed counter
CLOSE files
TERMINATE the program
```

An example of a pseudo code design specification

Structured flow charts

- Structured flow charts differ from traditional flowcharts in that structured flowcharts are restricted to compositions of certain basic forms.
- This makes the resulting flowchart the graphical equivalent of a structured pseudo code description.
- Flowcharts incorporate rectangular boxes for actions, diamond shaped boxes for decisions, directed arcs for specifying interconnection between boxes and a variety of specially shaped symbols to denote input, output, data stores etc.



```
S1
WHILE T1 DO
IF T2 THEN
S2
REPEAT
S3
IF T3 THEN
S4
ELSE
S5
END IF
UNITIL T4
ELSE
S7
END IF
S6
END WHILE
S8
```

Structured English

- Structured English can be used to provide a step by step specification for an algorithm. Like pseudo code, Structured English can be used at any desired level of detail. Structured English is often used to specify cookbook recepies.

- 1. Preheat oven to 350 degrees F
- 2. Mix eggs, milk and vanilla
- 3. Add flour and baking soda
- 4. Pour into a greased baking dish
- 5. Cook until done.

Decision Tables

- Decision tables can be used to specify complex decision logic in a high-level software specification. They are also useful for specifying algorithmic logic during detailed design

3.Explain briefly about Modules and modularization criteria

- **Modules and modularization criteria**
- Architectural design has the goal of producing well structured modular software systems. A software module to be a named entity having the following characteristics.

1. Modules contain instructions, processing logic and *data* structures
2. Modules can be separately, compiled and stored in a library
3. Modules can be included in a program.
4. Module segments can be used by invoking a name and some parameter.
5. Modules can use other module

Coupling

- A fundamental goal of software design is to structure the software product so that the number and complexity of inter connection between modules is minimized.

- The strength of coupling between two modules is influenced by the complexity of the interface, the type of connection and the type of communication

- Modification of a common data block or control block may require modifications of all routines that are coupled to that block.
- Connection established by referring to other module names are more loosely coupled than connections established by referring internal elements of other modules.
- The degree of coupling is lowest for data communication, higher for control communication and highest for modules that modify other modules.
- Coupling between modules can be ranked on a scale of strongest to weakest as follows.
- content coupling
- common coupling
- control coupling
- stamp coupling
- data coupling
- **Content coupling** occur when one module modifies local data values or instructions in another module. Content coupling can occur is assembly language programs.
- **Common coupling** module are bound together by global data structures. For instance, common coupling results when all routines in a FORTRAN program reference a single common data block. <,

- **Control coupling** involves passing control flags (as parameters or globals) between modules so that one module controls the sequence of processing steps in another module.
- **Stamp coupling** is similar to common coupling, except that global data items are shared selectively among routines that require the data**.**

Data coupling involves the use of parameter lists to pass data items between routines. The most desirable form of coupling between modules is a combination of stamp and data coupling.

Cohesion

- The internal cohesion of a module is measured in terms of the strength of binding of element with in the module
- . Cohesion of element occurs on the scale of weakest to strongest in the following order

1. coincidental cohesion**s**
2. logical cohesion
3. temporal cohesion
4. communication cohesion
5. sequential cohesion
6. functional cohesion
7. information cohesion.

Coincidental cohesion occurs when the elements within a module have no apparent relationship to one another.

Logical cohesion implies some relationship among the elements of the module, as for example, in a module that performs all input and output operations or in a module that edits all data.

Module with **temporal cohesion** exhibit many of the same advantages as logically bound modules.

The elements of a module possessing **communicational cohesion** refer to the same set of input and or output data for example, "print and punch the output file" is communicational bound.

Sequential cohesion of elements occurs when the output of one element is the input for the next element.

Functional cohesion is a strong and hence desirable type of binding of elements in a module because all elements are related to the performance of a single function.

Information cohesion of elements in a module occurs when module contains a complex data structure and several routines to manipulate the data structure.

Unit-IV&V

TWO MARKS

1.Define Unit testing

Unit testing comprises the set of tests performed by an individual programmer prior to integration of the unit into a larger system.

2. **Define Debugging**

Debugging is the process of isolating and correcting the causes of known errors. Success at debugging requires highly developed problem-solving skills; Commonly used debugging methods include induction, deduction, and backtracking.

3.Define System testing

System testing involves two kinds of activities: integration testing and acceptance testing. Strategies for integrating software components into a functioning product include the bottom-up strategy, the top-down strategy, and the sandwich strategy.

4.What are the advantages of Top-down integration

Top-down integration offers several advantages:

1. System integration is distributed throughout the implementation phase. Modules are integrated as they are developed.

2. Top-level interfaces are tested first and most often.

3. The top-level routines provide a natural test harness. for lower-level routines.

4: Errors are localized to the new modules and interfaces that are being added.

5.Define Acceptance testing

Acceptance testing involves planning and execution of functional tests, performance tests, and stress tests to verify that the implemented system satisfies its requirements.

6.What are the disadvantages are Maintenance activities

- **The disadvantages are:** Maintenance activities may divert the developers from their new project. Maintenance by the developers make the maintenance activity vulnerable to personnel turnover.
- Maintenance by a separate group forces more attention to standards and high quality documentation.

TEN MARKS

1.Discuss in detail about system testing

- System testing involves two kinds of activities: integration testing and acceptance testing.
- Strategies for integrating software components into a functioning product include the bottom-up strategy, the top-down strategy, and the sandwich strategy.
- The integration strategy dictates the order in which modules must be available, and thus exerts a strong influence on the order in which modules are written, debugged, and unit tested.

Acceptance testing

Acceptance testing involves planning and execution of functional tests, performance tests, and stress tests to verify that the implemented system satisfies its requirements.

Integration Testing

- Bottom-up integration is the traditional strategy used to integrate the components of a software system into a functioning whole.
- Bottom-up integration consists of unit testing, followed by subsystem testing, followed by testing of the" entire system. A subsystem consists of several modules that communicate with each other through well-defined interfaces.
- System testing is concerned with subtleties in the interfaces, decision logic, % control flow, recovery procedures, throughput, capacity, and timing characteristics of the entire system.
- Disadvantages of bottom-up testing include the necessity to write and debug test harnesses for the modules and subsystems, and the level of complexity that results from combining modules and subsystems into larger and larger units..
- Top-down integration starts with the main routine and one or two immediately subordinate routines in the system structure.
- Top-down integration requires the use of program stubs to simulate the effect of lower-level routines that are called by those being tested.

2.Explain in detail about Managerial aspects of software maintenance.

Managerial aspects of software maintenance

- Successful s/w product requires a combination of managerial skills and technical expertise.

- One of the most important aspects of s/w maintenance involves tracking and control of maintenance activities.
- Maintenance activity for a s/w product usually occurs in response to change request filed by the user of the product. Change request processing; can be described by the following algorithm.
- -software change request initiated.
- -request analyzed
- if (change control board concurs) then
- -modifications performed with priority and constraints established by change
- control board.
- -regression tests performed -changes submitted to change control boards. if (change control board approves) then -master tape updated -external documentation updated -update distributed as directed by change control.
- **Change Control board**
- This reviews and approves change requests. Approved changes are forwarded to the maintenance programmers for action. The s/w is modified, revalidated and submitted to the change
- **Change request Summaries**
- The status of change requests and s/w maintenance activities should be summarized on a weekly or monthly basis. The summary should report emergency problems.
- **Quality assurance activities**

- The primary function of a quality assurance group during s/w maintenance is to ensure the s/w quality does not degrade as a result of maintenance activities.
- **Organizing maintenance programmers**
- Software maintenance can be performed by the development team or by members of a separate organization.
- **The advantages of development team are**: Members of the development team will be intimately familiar with the product.
- They will take great care to design and implement the system to enhance maintainability.
- **The disadvantages are:** Maintenance activities may divert the developers from their new project. Maintenance by the developers make the maintenance activity vulnerable to personnel turnover.
- Maintenance by a separate group forces more attention to standards and high quality documentation.

3.Discuss about walkthroughs and inspections

- A structured walkthrough is an in depth technical review of some aspect of a software system.
- Walkthrough can be used at any time, during any phase of a software project.
- A walkthrough team-consists of four to six people. The person whose material is being reviewed is responsible for providing copies of the review material to members of the walkthrough team in advance of the walkthrough session, and team members are responsible for reviewing the material prior to the session.

- It is an important to emphasize that the material is reviewed, and not the reviews, the focus of a walkthrough is on detection of errors and not on corrective actions.

Design Guidelines

- There is no magic formula or recipe for software design. Design is a creative process that can be guided and directed, but it can never be reduced to an algorithmic procedure.
- Review the requirements specification. In particular, study the desired functional characteristics and performance attributes of the system.
- Review and expand the external interfaces, user dialogues, and report formats developed during requirements analysis.
- Review and refine the data flow diagrams developed during requirements analysis and external design. Identify internal data stores and elaborate the processing functions.
- Identify functional abstractions and data abstractions. Record them using a design notation.
- Define the visible interfaces for each functional abstraction and each data abstraction. Record them using a design notation.
- Define the modularization criteria to be used in establishing system structure.
- Apply the techniques of your particular design method to establish system structure.
- Iterate steps 4 through 7 and as necessary, steps 1 to 7 until a suitable structure is achieved.
- Verify that the resulting system structure satisfies the requirements.

- Develop interface specifications for the procedures in each module. Conduct the Preliminary Design Review. The level of detail should be inclusive to
- levels 1 and 2 of the procedure templates.
- Develop concrete data representations for the data stores and data abstractions.
- Expand the procedure templates to include the information in level 3.
- Specify algorithmic details for the body of each procedure in the system, using successive refinement. HIPO diagrams, pseudocode, Structured English and / or structured flow charts. Develop concrete data representation and inter connections between data structures and algorithms.
- Conduct the Critical Design Review.
- Redesign as necessary.) The major difficulties in software design are caused by inadequate requirements.

FIVE MARKS

1. Discuss about the Formal verification

- Formal verification involves the use of rigorous, mathematical techniques to demonstrate that computer programs have certain desired properties.
- The methods of input-output assertions, weakest preconditions, and structural induction are three commonly used techniques.
- Using input-output assertions, predicates (assertions) are associated with the entry point, the exit point, and various intermediate points in the source code.

- The predicates, or verification conditions, must be true whenever the associated code is executed.
- The notation (P) S (R) is used to mean that if predicate P is true prior to executing code segment S, predicate R will be true following execution of S
- Termination is proved by showing that the execution sequence for each loop monotonically decreases (increases) some nonnegative (negative) property on each pass through the loop.

2.Explain about Other maintenance tools and techniques

- Automated tools to support software maintenance include technical support tools and managerial tools to support the technical aspects of software maintenance include analysis and design tools, implementation tools and debugging and testing tools.
- Automated tools of special importance for software maintenance include text editors, debugging aids, cross-reference generators, linkage editors, comparators, complexity metric calculator, version
- control systems and configuration management data bases.
- Test editors can be used to insert and replace segments source code,
- internal comments, test data ad supporting documents to systematically change all

occurrences of an identifiers or other textual string, to locate all references to a given identifier or other string of text and to save both old and new versions of a routine, test file of document.

- Debugging aids provide traps, dumps, traces, assertion checking and history files to aid in locating the causes of known errors.

System level cross-reference generators provide cross-reference listings for procedure calls, statement usage and data references.

- Cross reference directories provide the calling structure the 'procedure names and statement numbers where formal parameters, local variables and global variables are defined, set and used.
- A linkage editor links together object modules of compiled code to produce an executable program. These can be used by a maintenance programmer to configure a system in various ways and to link selectivity recompiled modules into a software system.
- A comparator compares tv.o files of information and reports the differences. Comparators can be used during maintenance to compare/two versions of source program, a test suite, a file of test results or two versions of a supporting documents. This allows to pinpoint the differences between versions, to determine whether a modification has achieved a desired result and to determine whether adverse side effects have been introduced by a modification

3. **Discuss about QUALITY ASSURANCE**

QUALITY ASSURANCE

- Quality assurance is "a planned and systematic pattern of all actions necessary to provide adequate confidence that the item or product conforms to established technical requirements".
- The purpose of a software quality assurance group is to provide assurance that the procedures, tools, and techniques used during product development and modification are adequate to provide the desired level of confidence in the work products.

- Often, software quality assurance personnel are organizationally distinct from the software development group.
- This adds a degree of impartiality to quality assurance activities, and allows quality assurance personnel to become specialists in their discipline.
- Preparation of a Software Quality Assurance Plan for each software project is a primary responsibility of the software quality assurance group.
- Topics in a Software Quality Assurance Plan include
- Purpose and scope of the plan
- Documents referenced in the plan
- Organizational structure, tasks to be performed, and specific responsibilities as they relate to product quality.
- Documents to be prepared and checks to be made for adequacy of the documentation
- Standards, practices, and conventions to be used

. Reviews and audits to be conducted

A configuration management plan that identifies software product items, controls and implements changes, and records and reports changed status

Practices and procedures to be followed in, reporting, tracking, and resolving software problems .

Specific tools and techniques to be used to support quality assurance activities

Methods and facilities to be used to maintain and store controlled versions of identified software

Methods and facilities to be used to protect computer program physical media

Provisions for ensuring the quality of vendor-provided and subcontractor-developed software

Methods and facilities to be used in collecting, maintaining, and retaining quality assurance records

Other duties performed by quality assurance personnel include:

1. Development of standard policies, practices, and procedures
2. Development of testing tools and other quality assurance aids
3. Performance of the quality assurance functions described in the Software Quality Assurance Plan for each project
4. Performance and documentation of final product acceptance tests for each software product.

Two marks

1.Write the classifications of documentation.

Types of documentation: - in broad of view the documentation can be classified in two types

a) internal documentation

b) external documentation

2.What are the types of external documentation?

In external documentation it can be again directed in 7 parts:

1) Management Documentation

2) System Documentation

3) Operational Documentation

4) User Documentation

5) Program Documentation

6) Training Documentation

7) Implementation Documentation

3.Define internal documentation.

Internal documentation refers the internal work process like coding, programming, structure which helps software analyst and programmer.

4.Define External documentation

The external documentation deals with the user manual and other related information for particular plan (customer).

5.Write the classifications of documentation.

Types of documentation: - in broad of view the documentation can be classified in two types-

a) internal documentation

b) external documentation

6.What are the types of external documentation?

In external documentation it can be again directed in 7 parts:

1) Management Documentation

2) System Documentation

3) Operational Documentation

4) User Documentation

5) Program Documentation

6) Training Documentation

7) Implementation Documentation

7.Define internal documentation.

Internal documentation refers the internal work process like coding, programming, structure which helps software analyst and programmer.

The external documentation deals with the user manual and other related information for particular plan (customer).

.

FIVE MARKS

1.Explain about system documentation

System documentation: - it has the following features for its own its:

1) System overview

2) SRS

3) Specification/design/implementation

4) Test plan

5) Data dictionary

6) Acceptance of test plan.

1) **System overview**: - it is very important step for making particular software for the software developer.

2) **SRS**: - it is important features for the system documentation. It has some specific goals which help for making particular software.

3) **Specification/design/implementation**: - the part deals with the system analyst that describe how to use, how to implement, how to make and how to make maintain the particular software including design phase.

4) **Test plan**: - it refers the hole works must be tested with some specific steps and parameters for deciding correctness for a particular work.

5) **Data dictionary**: - it contain data about data (Meta data) this data's are useful for implementation from the coder side.

6) **Acceptance of test plan**: - this feature include in the system documentation part to know whether the design implementation and progress is accepted by designer developer and some time for user.

user.

2.Explain about principles of system documentation.

1) Availability

2) Objectivity

3) Cross Referencing

4) Easy To Maintain

5) Completeness

1) **Availability**: - it refers the documentation should be available for the analyst, developer and the user.

2) **Objectivity**: - the objectivity must be focused and clear who are making the documentation for its specific use. Objectivity refers the original method for making particular software.

3) **Cross referencing**: - it determines for internal communication between to or more module. This feature includes the internal relational ship between the modules.

4) **Easy to maintain**: - after complete the documentation some times it refers in future it may be changed at this time the documentation should be worked properly and correctly.

5) **Completeness**: - it refers the all phase including design, coding, testing, user manual. This must be included for making the documentation successful and complete.

TEN MARKS

1.Explain in detail about Documentation.

Documentation is a serial part in the S.E method of for making its own success for internal and external users.

Types of documentation: - in broad of view the documentation can be classified in two types-

a) Internal Documentation

b) External Documentation

In external documentation it can be again directed in 7 parts:

1) Management Documentation

2) System Documentation

3) Operational Documentation

4) User Documentation

5) Program Documentation

6) Training Documentation

7) Implementation Documentation

Internal documentation refers the internal work process like coding, programming, structure which helps software analyst and programmer.

The external documentation deals with the user manual and other related information for particular plan (customer).

1) **User documentation**: **-** it refers the whole user details for particular software and something it refers how the are going on in a particular software.

2) **Programming**: **-** these types of documentation help the programmer and other system level manager to directly or indirectly involve with programming structure.

3) **System:** - the goals of to focus on the software as well as hardware component in which particular software run.

4) **Management**: - it is used for the upper level manager and its contains the whole work project plan including time and cost for a software.

5) **Operational**: - it maintains the internal working process including some problem solution for the customer. 6) Training: - it basically is a summarized from for training purpose and user guideline of a software.

7) **Implementation**: - it generally helps to internal process. It describes how the software will be implementing in a particular system.

System documentation: - it has the following features for its own its:

1) System overview

2) SRS

3) Specification/design/implementation

4) Test plan

5) Data dictionary

6) Acceptance of test plan.

1) **System overview**: - it is very important step for making particular software for the software developer.

2) **SRS**: - it is important features for the system documentation. It has some specific goals which help for making particular software.

3) **Specification/design/implementation**: - the part deals with the system analyst that describe how to use, how to implement, how to make and how to make maintain the particular software including design phase.

4**) Test plan**: - it refers the hole works must be tested with some specific steps and parameters for deciding correctness for a particular work.

5) **Data dictionary**: - it contain data about data (Meta data) this data's are useful for implementation from the coder side.

6) **Acceptance of test plan**: **-** this feature include in the system documentation part to know whether the design implementation and progress is accepted by designer developer and some time for user.