Biyani's Think Tank

Concept based notes

Software Engineering

BCA Part-III

Pallavi Gore

Revised By: Kritika Saxena

Deptt. of Information Technology

Biyani Girls College, Jaipur

Published by :
Think Tanks
Biyani Group of Colleges

Concept & Copyright :
©Biyani Shikshan Samiti
Sector-3, Vidhyadhar Nagar,
Jaipur-302 023 (Rajasthan)
Ph : 0141-2338371, 2338591-95 • Fax : 0141-2338007
E-mail : acad@biyanicolleges.org
Website :www.gurukpo.com; www.biyanicolleges.org

ISBN: 978-93-82801-77-1

First Edition : 2009
Second Edition: 2010
Price:

Leaser Type Setted by :
Biyani College Printing Department

Preface

I am glad to present this book, especially designed to serve the needs of the students. The book has been written keeping in mind the general weakness in understanding the fundamental concepts of the topics. The book is self-explanatory and adopts the "Teach Yourself" style. It is based on question-answer pattern. The language of book is quite easy and understandable based on scientific approach.

Any further improvement in the contents of the book by making corrections, omission and inclusion is keen to be achieved based on suggestions from the readers for which the author shall be obliged.

I acknowledge special thanks to Mr. Rajeev Biyani, *Chairman* & Dr. Sanjay Biyani, *Director* (*Acad.*) Biyani Group of Colleges, who are the backbones and main concept provider and also have been constant source of motivation throughout this Endeavour. They played an active role in coordinating the various stages of this Endeavour and spearheaded the publishing work.

I look forward to receiving valuable suggestions from professors of various educational institutions, other faculty members and students for improvement of the quality of the book. The reader may feel free to send in their comments and suggestions to the under mentioned address.

Author

Syllabus

BCA

Software Engineering

Software Characteristics, Components, Applications, Software process Models: Waterfall, spiral, Prototyping, Fourth Generation Techniques, Concepts of Project Management, and Role of Metrics & Measurements.
S/W Project planning Objectives, Decomposition techniques: S/W Sizing, Problem-based estimation, Process based estimation, Cost Estimation Models: COCOMO Model.
S/W Design: Objectives, Principles, Concepts, Design methodologies Data design, Architectural design, procedural design, Object oriented concepts
Testing fundamentals: Objectives, principles, testability, Test cases: White box & Black box testing strategies: verification & validation, unit test, integration testing, validation, testing, system testing

Content

3.	**Software Project Management** 3.1 Charters tics of s/w Product 3.2 Project table 3.3 Time Box 3.4 Decomposition technique 3.5 Software sizing and size Metrics 3.6 Cost estimation 3.7 Cost factor 3.8 COCOMO Model 3.9 Time line chart 3.10 Project Monitoring 3.11 Project control 3.12 Cost Control

Chapter-1

Introduction to Software Engineering

Q.1 Define the term Software.

Ans.: In general, software can be defined as a collection of computer programs, which in turn is a collection of commands.

But there is a distinction between a program & a programming system product. Unlike a program which is normally used by its author, a product requires documentation to help users which are other than the developers of the system.

To conclude we can define software as the collection of computer programs, procedures, rules & associated documents & data.

Q.2 what is Software Engineering?

Ans.: Software Engineering is the discipline that aims to provide methods & procedures for developing software system.

It is the application of a systematic disciplined & quantifiable approach of development & maintenance of software. It includes different techniques & procedures of "Software development process to improve the reliability of software".

In other words, software Engineering is the application of science & maths by which the capabilities of computer equipments are made useful to man via computer programs.

Q 3. Explain the Software Characteristics?

Ans. The key Characteristics of Software are as follows-

1. **Most software is custom build, rather than being assembled from existing components**:- Most software continue to be custom built, although recent developments tend to component based. Modern reusable components encapsulate both data and the processing applied to data, enabling the software engineer to create new applications from reusable part.

2. **Software is developed or engineered; it is not manufactured in the classical sense-**Although some similarities exist between software development and hardware manufactured, the two activities are fundamentally different. In both activities, high quality is achieved through good design, but the manufacturing phase for hardware can introduce quality problems that are nonexistent for software. Both activities depend on people, but the relationship between people applied and work accomplished is entirely different. Both require the construction of a "product". But the approaches are different.

3. **Software is flexible-** A program can be developed to do almost anything. sometimes, this characteristics may be the best and may help us to accommodate any kind of change.
4. **Software does not wear out-** There is a well known "Bath-tub curve" in reliability studies for the hardware product.
   There are three phases for the life of a hardware product. Initial phase is burn-in phase, where failure intensity is high. It is expected to test the product in the industry before delivery. Due to testing and fixing faults, failure intensity will come down initially and may stabilize after certain time. The second phase is the useful life phase where failure intensity is approximately constant and is called useful life of a product. After few years, again failure intensity will increase due to wearing out of components. This phase is called wear out phase.

Q 4. Define the Software Components?

Ans. A software component is a system element offering a predefined service and able to communicate with other components. Clemens Szyperski and David Messerschmitt give the following five criteria for what a software component shall be fulfill the definition:

1. Multiple-use
2. Non-context-specific
3. Composable with other components
4. Encapsulated i.e. ,non-investigable through its interfaces
5. A unit of independent and versioning.

A simpler definition can be: a component is an object written to a specification. It does not matter what the specification is; COM, JavaBeans etc, as long as the object adhere to the specification.

It takes significant effort and awareness to write a software component that effectively reusable. The component needs:

- To be fully documented ;
- More through testing;

- Robust input validity checking
- To pass back useful error messages as appropriate;
- To be build with an awareness that it will be put to unfore seen uses;
- A mechanism for compensating developers who invest the effort implied above

Q 5. Define the Software Applications?

Ans. Software applications are grouped into eight areas for convenience as shown in fig.

1. **System Software**- System Software is a collection of programs used to run the system as an assistance to use other software programs. The compilers, editors, utilities operating system components, drivers and interfaces are examples of system software.
2. **Real-time Software**- Real-time Software deals with changing environment. First it collect the input and convert it from analog to digital, control component that responds to the external environment, perform the action in the last.
3. **Embedded Software**- Software, when written to perform certain functions under control conditions and further embedded into hardware as a part of large systems, is called Embedded Software.
4. **Business Software**- Software designed to process business applications is called business software. Business software could be a data and information-processing application.
5. **Personal Computer Software**- Word processing, spread sheets, computer graphics, multimedia, entertainment, database management, personal and business financial applications, external network or database access are only a few of hundreds of applications.
6. **Artificial Intelligence Software**- Artificial Intelligence Software uses non-numerical algorithms, which use the data and information generated in the system, to solve the solving problems.
   Applications within this area include robotics, expert system, pattern recognition, artificial neural networks, theorem proving and game playing, signal processing software.
7. **Web-Based Software-** Web-Based Software is the browsers by which web pages are processed i.e., HTML, Java, CGI, Perl, DHTML etc.
8. **Engineering and Scientific Software-** Design, engineering of scientific software's deal with processing requirements in their specific fields. They are written for specific applications using the principles and formulas of each field.
   These software's service the need of drawing, drafting, modeling, lead calculations, specifications-building and so on. Dedicated software's are available for stress analysis or for analysis of engineering data, statistical

data for interpretation and decision-making. CAD/CAM/CAE packages, SPSS, MATLAB, Circuit analyzers are typical examples of such software.

Q.6 What is Software Development Life Cycle?

OR

Explain in detail the different stages of SDLC.

Ans.: It is a concept of providing a complete support to a software product throughout all the stages of its evolution

The systems development life cycle (SDLC) is a conceptual model used in project_management that describes the stages involved in an information system development project, from an initial feasibility study to maintenance of the completed application.

The SDLC Waterfall Model :

Small to medium database software projects are generally broken down into six stages:

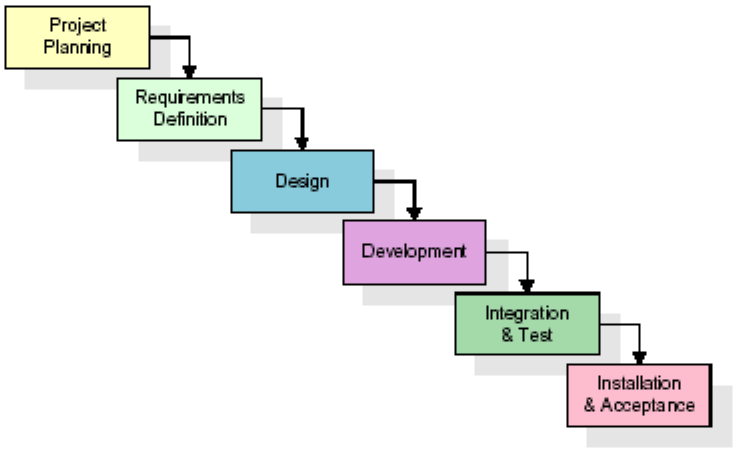


Diagram : Waterfall Model

Here the outputs from a specific stage serve as the initial inputs for the following stage.

(1) Planning Stage :

The planning stage establishes a bird's eye view of the software product, and uses this to establish the basic project structure. This stage is used to evaluate feasibility and risks associated with the project, and describe appropriate management and technical approaches.

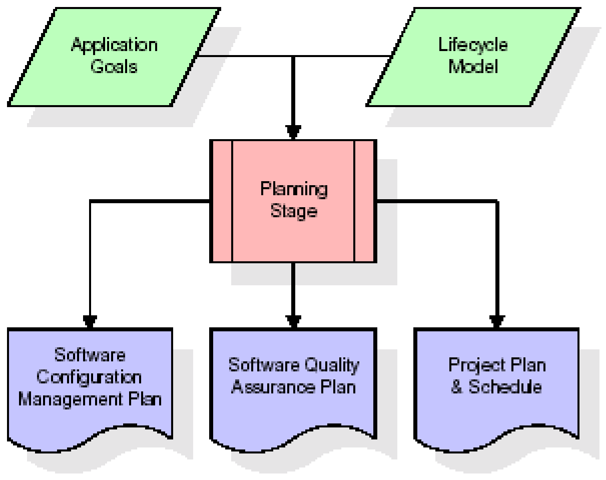

Application Goals
Lifecycle Model
Planning Stage
Software Configuration Management Plan
Software Quality Assurance Plan
Project Plan & Schedule

Diagram : Planning

The most critical section of the project plan is a listing of high-level product requirements, also referred to as goals. All the software product requirements to be developed during the requirements definition stage flow from one or more of these goals. The minimum information for each goal consists of a title and textual description, although additional information and references to external documents may be included.

The outputs of the project planning stage are the configuration management plan, the quality assurance plan, and the project plan and schedule.

(2) Requirements Definition Stage :

The requirements gathering process takes as its input the goals identified in the project plan. Each goal is refined into a set of one or more requirements. These requirements define the major functions of the application, operational data areas and reference data areas. Each of these definitions is termed a Requirement. Requirements are identified by unique requirement identifiers and contain a requirement title and textual description.

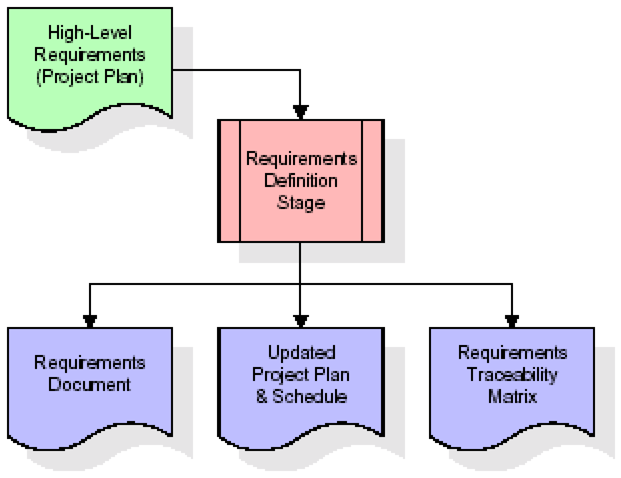

High-Level Requirements (Project Plan)
Requirements Definition Stage
Requirements Document
Updated Project Plan & Schedule
Requirements Traceability Matrix

Diagram : Requirement – Definition

These requirements are fully described in the Requirements Document and the Requirements Traceability Matrix (RTM). The requirements document contains complete description of each requirement, including diagrams and references to external documents as necessary.

Note : detailed listings of database tables and fields are *not* included in the requirements document.

The outputs of the requirements definition stage include :

i) the requirements document

ii) the RTM

iii) an updated project plan.

(3) Design Stage :

The design stage takes as its initial input the requirements identified in the approved requirements document. Design elements describe the desired software features in detail, and generally include functional hierarchy diagrams, screen layout diagrams, tables of business rules, business process diagrams, pseudo code, and a complete entity-relationship diagram with a full data dictionary. These design elements are intended to describe the software in sufficient detail so that the skilled programmers may develop the software with minimal additional input.

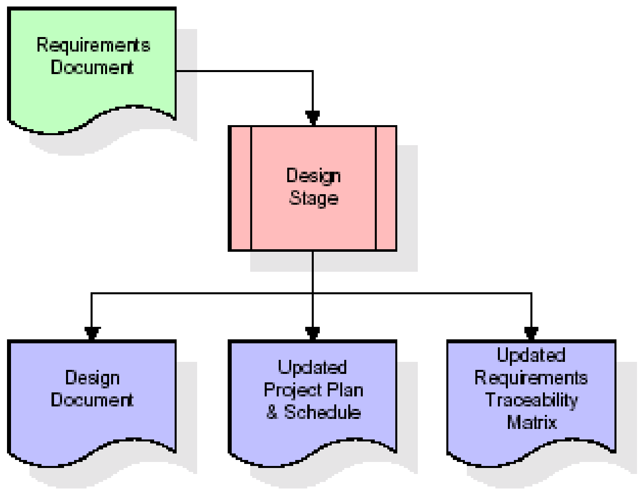

Requirements Document
Design Stage
Design Document
Updated Project Plan & Schedule
Updated Requirements Traceability Matrix

Diagram : Design

The outputs of the design stage :

i) the design document

ii) an updated RTM,

iii) updated project plan.

(4) Development Stage :

Here for each design element, a set of one or more software artifacts will be produced. Software artifacts include but are not limited to menus, data management forms, data reporting formats, and specialized procedures and functions. Appropriate test cases will be developed for each set of functionally related software artifacts, and an online help system will be developed to guide users in their interactions with the software.

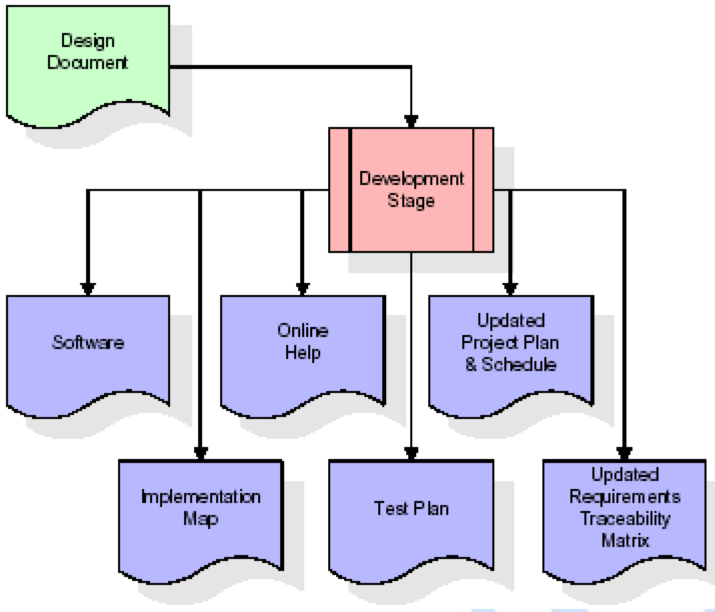


Diagram : Development

The outputs of the development stage include:

i) A fully functional set of software that satisfies the requirements and design elements previously documented.

ii) An online help system that describes the operation of the software.

iii) An implementation map that identifies the primary code entry points for all major system functions.

iv) A test plan that describes the test cases to be used to validate the correctness and completeness of the software.

v) An updated RTM.

vi) An updated project plan.

(5) Integration & Test Stage:

During the integration and test stage, the software artifacts, online help, and test data are migrated from the development environment to a separate test environment. At this point, all test cases are run to verify the correctness and completeness of the software. Successful execution of the test module confirms a robust and complete migration capability. During this stage, reference data is finalized for production use and production users are identified and linked to their appropriate roles. The final reference data (or links to reference data source files) and production user lists are compiled into the Production Initiation Plan.

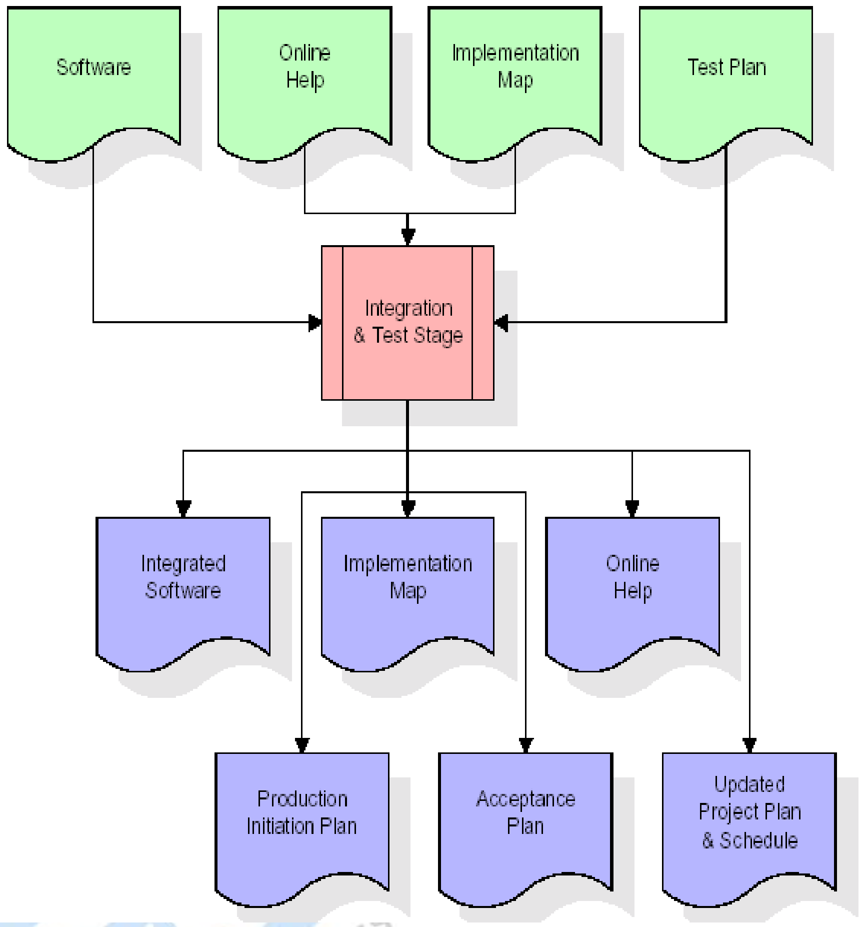


Diagram: Integration & Testing

The outputs of the integration and test stage include:

i) An integrated set of software.

ii) An online help system.

iii) An implementation map.

iv) A production initiation plan that describes reference data and production users.

v) An acceptance plan which contains the final module of test cases.

vi) An updated project plan.

(6) Installation & Acceptance Stage :

During the installation and acceptance stage, the software artifacts, online help, and initial production data are loaded onto the production server, all test cases are run to verify the correctness and completeness of the software. After customer personnel have verified that the initial production data load is correct and the test suite has been executed with satisfactory results, the customer formally accepts the delivery of the software.

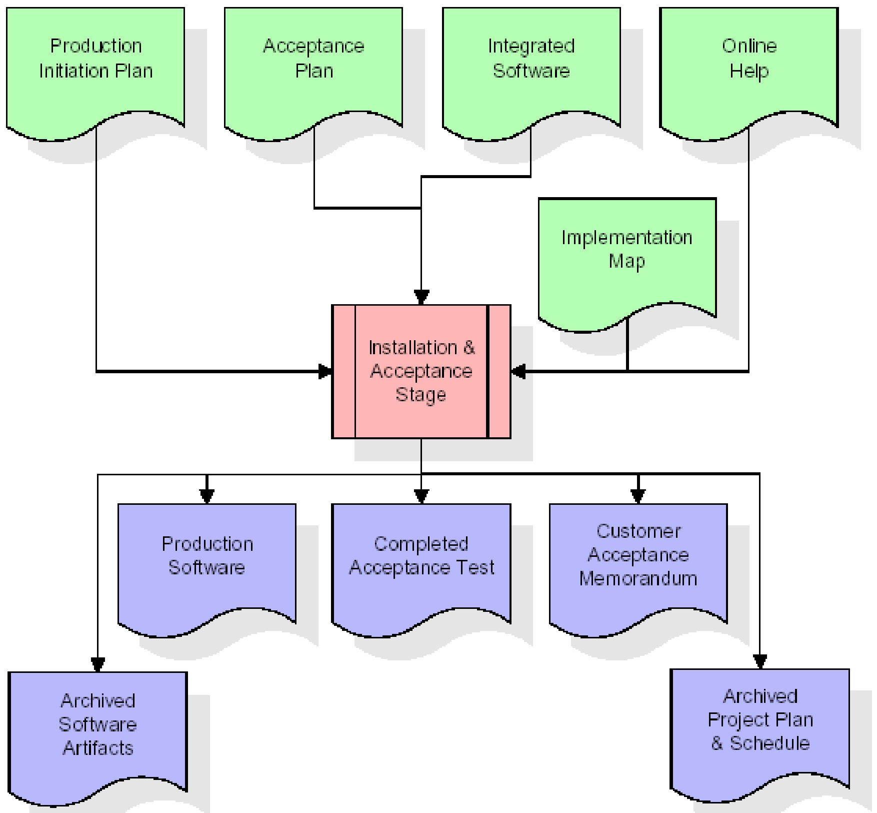

Production Initiation Plan
Acceptance Plan
Integrated Software
Online Help
Implementation Map
Installation & Acceptance Stage
Production Software
Completed Acceptance Test
Customer Acceptance Memorandum
Archived Software Artifacts
Archived Project Plan & Schedule

Diagram : Installation & Acceptance

The primary outputs of the installation and acceptance stage include:

i) A production application.

ii) A completed acceptance test suite.

iii) A memorandum of customer acceptance of the software.

Q.7 Define prototyping model.

Ans.: The prototyping model begins with the requirements gathering. The developer and the customer meet and define the objectives for the software, identify the needs, etc. A 'quick design' is then created. This design focuses on those aspects of the software that will be visible to the customer. It then leads to the construction of a prototype. The prototype is then checked by the customer and any modifications or changes that are required are made to the prototype. Looping takes place in this process and better versions of the prototype are created. These are continuously shown to the user so that any new changes can be updated in the prototype. This process continues till the user is satisfied with the system. Once a user is satisfied, the prototype is converted to the actual system with all considerations for quality and security.

The prototype is considered as the 'first system'. It is advantageous because both the customers and the developers get a feel of the actual system. But there are certain problems with the prototyping model too.

i) The prototype is usually created without taking into consideration overall software quality.

ii) When the customer sees a working model in the form of a prototype, and then is told that the actual software is not created, the customer can get irritated.

iii) Since the prototype is to be created quickly, the developer will use whatever choices he has at that particular time e.g. he may not know a good programming language, but later may learn. He then cannot change the whole system for the new programming language). Thus the prototype may be created with less-than-ideal choices.

This model may be one of the following types :

1. **Throwaway model**: discard the model once all requirements are understood.
2. **Evolving Model:** Refine the model every time when the requirements are clear.

Q.8 Define Spiral model?

Ans. The spiral model, originally proposed by Boehm, is evolutionary software Process model that couples the iterative nature of prototyping with the controlled and systematic aspects of the linear sequential model. It provides the potential for rapid development of incremental versions of the software. Using the spiral model, software is developed in a series of incremental releases. During early iterations, the incremental release might be a paper model or prototype. During later iterations, increasingly more complete versions of the engineered system are produced.

A spiral model is divided into a number of framework activities, also called task Regions 6. Typically, there are between three and six task regions.

- **Customer communication**—tasks required to establish effective communication between developer and customer.

- **Planning**—tasks required to define resources, timelines, and other project related information.
- **Risk analysis**—tasks required to assess both technical and management risks.
- **Engineering**—tasks required to build one or more representations of the application.
- **Construction and release**—tasks required to construct, test, install, and provide user support (e.g., documentation and training).

Q 9. What is Forth Generation Techniques?

Ans. The term fourth generation technique (4GT) encompasses a broad array of software tools that have one thing in common: each enables the software engineer to specify some characteristic of software at a high level. The tool then automatically generates source code based on the developer's specification. There is little debate that the higher the level at which software can be specified to a machine, the faster a program can be built. The 4GT paradigm for software engineering focuses on the ability to specify software using specialized language forms or a graphic notation that describes the problem to be solved in terms that the customer can understand.

Currently, a software development environment that supports the 4GT paradigm includes some or all of the following tools: nonprocedural languages for database query, report generation, data manipulation, screen interaction and definition, code generation; high-level graphics capability; spreadsheet capability, and automated generation of HTML and similar languages used for Web-site creation using advanced software tools. Initially, many of the tools noted previously were available only for very specific application domains, but today 4GT environments have been extended to address most software application categories.

For small applications, it may be possible to move directly from the requirements gathering step to implementation using a nonprocedural fourth generation language (4GL) or a model composed of a network of graphical icons. However, for larger efforts, it is necessary to develop a design strategy for the system, even if a 4GL is to be used. The use of 4GT without design (for large projects) will cause the same difficulties (poor quality, poor maintainability, poor customer acceptance) that have been encountered when developing software using conventional approaches.

Implementation using a 4GL enables the software developer to represent desired results in a manner that leads to automatic generation of code to create those results.

Obviously, a data structure with relevant information must exist and be readily accessible by the 4GL.

To transform a 4GT implementation into a product, the developer must conduct thorough testing, develop meaningful documentation, and perform all other solution integration activities that are required in other software engineering paradigms. In addition, the 4GT developed software must be built in a manner that enables maintenance to be performed expeditiously.
Like all software engineering paradigms, the 4GT model has advantages and disadvantages.
Proponents claim dramatic reduction in software development time and greatly improved productivity for people who build software. Opponents claim that current 4GT tools are not all that much easier to use than programming languages, that the resultant source code produced by such tools is "inefficient," and that the maintainability of large software systems developed using 4GT is open to question.
There is some merit in the claims of both sides and it is possible to summarize the current state of 4GT approaches:

1. The use of 4GT is a viable approach for many different application areas.
Coupled with computer-aided software engineering tools and code generators, 4GT offers a credible solution to many software problems.

2. Data collected from companies that use 4GT indicate that the time required to produce software is greatly reduced for small and intermediate applications and that the amount of design and analysis for small applications is also reduced.

3. However, the use of 4GT for large software development efforts demands as much or more analysis, design, and testing (software engineering activities) to achieve substantial time savings that result from the elimination of coding.

To summarize, fourth generation techniques have already become an important part of software engineering. When coupled with component-based development approaches the 4GT paradigm may become the dominant approach to software development.

Q.10 Write a short note on Knowledge Engineering.

Ans.: **Knowledge Engineering (KE)** refers to the building, maintaining and development of knowledge-based systems. It has a great deal in common with software engineering, and is related to many computer science domains such as artificial intelligence, databases, data mining, expert systems, decision support systems and geographic information systems. Knowledge engineering is also related to mathematical logic and is structured according to our understanding of how human reasoning and logic works.

Various activities of KE specific for the development of a knowledge-based system are :

- Assessment of the problem
- Development of a knowledge-based system shell / structure
- Implementation of the structured knowledge into knowledge –bases
- Acquisition and structuring of the related information, knowledge and specific preferences
- Testing and validation of the inserted knowledge
- Integration and maintenance of the system
- Revision and evaluation of the system

Knowledge Engineering Principles: - Since the mid – 1980s, knowledge engineers have developed a number of principles, methods and tools that considerably improved the process of knowledge acquisition and ordering. Some of the key principles are summarized as follows:

- Knowledge engineers acknowledge that there are different types of knowledge and that the right approach and technique should be used for the knowledge required.
- Knowledge engineers acknowledge that there are different types of experts and expertise, and therefore the methods should be chosen appropriately.
- Knowledge engineers recognize that there are different ways of representing knowledge, which can aid the acquisition, validation and re-use of knowledge.
- Knowledge engineers recognize that there are different ways of using knowledge, so that the acquisition process can be guided by the project aims (goal-oriented).
- Knowledge engineers use structured methods to increase the efficiency of the acquisition process.

Views of Knowledge Engineering: - There are two main views to knowledge engineering:

- **Transfer View –** This is the traditional view. In this view, we apply conventional knowledge engineering techniques to transfer human knowledge into artificial intelligence systems.

- **Modeling View –** This is the alternative view. In this view, the knowledge engineer attempts to model the knowledge and problem solving techniques of the domain expert into the artificial intelligent system.

Q.11 what is End User Development?

Ans.: End User Development (EUD) activities range from customization to component configuration and programming.

In scientific and engineering domains end users frequently develop complex systems with standard programming languages such as C++ and JAVA. However only a minority of users adapt COTS (Customer Off The Shelf software) products; furthermore, composing systems from reusable components, such as ERP (Enterprise Resource Plans) systems defeat most end users, who resort to expensive and scarce expert developers for implementation. So EUD is only a partial success story.

End User development depends on a fine balance between user motivation, effective tools and management support.

EUD tools and technology: Design of language for user-computer communication pose a conflict between complexity and power. More complex languages can address a wider range of problems but impose an increasing learning burden on the user.

The goal for EUD tools is to reduce the learning burden while providing powerful facilities to address a wide range of problems. Given that some learning burden will always be present, EUD tools need to motivate their users.

□ □ □

Chapter-2

Project Management

Q 1. Define the concept of project management?

Ans. Software project management refers to manage the complete software project. It is the means by which an orderly control process can be imposed on the software development process in order to ensure software quality.

In order to conduct a successful project, we must understand the scope of the work to be done, the risks to be incurred, the resources to be required, the task to be accomplished, the milestones to be tracked, the effort (cost) to be expanded and the schedule to be followed.

Effective software project management focuses on the four P's: **people, product, process, and project**.

(1) People- The "people factor" is so important that the Software Engineering Institute has developed a *people management capability maturity model* (PM-CMM), "to enhance the readiness of software organizations to undertake increasingly complex applications by helping to attract, grow, motivate, deploy, and retain the talent needed to improve their software development capability".

Now we examine the players who participate in the software process and the manner in which they are organized to perform effective software engineering

(i) The Players-

The software process (and every software project) is populated by players who can be categorized into one of five constituencies:

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

Every software project is populated by people who fall within this taxonomy. To be effective, the project team must be organized in a way that maximizes each person's skills and abilities. And that's the job of the team leader.

(ii) Team Leaders

- **Motivation.** The ability to encourage (by "push or pull") technical people to produce to their best ability.
- **Organization.** The ability to mold existing processes (or invent new ones) that will enable the initial concept to be translated into a final product.
- **Ideas or innovation.** The ability to encourage people to create and feel creative even when they must work within bounds established for particular software product or application.

(iii) The Software Team

1. *n* individuals are assigned to *m* different functional tasks, relatively little combined work occurs; coordination is the responsibility of a software manager who may have six other projects to be concerned with.

2. *n* individuals are assigned to *m* different functional tasks ($m < n$) so that informal "teams" are established; an ad hoc team leader may be appointed; coordination among teams is the responsibility of a software manager.
   Mantei [MAN81] suggests three generic team organizations:

Democratic decentralized (DD). This software engineering team has no permanent
Leader. Rather, "task coordinators are appointed for short durations and
Then replaced by others who may coordinate different tasks." Decisions on problems and approach are made by group consensus. Communication among team
Members are horizontal.

Controlled decentralized (CD). This software engineering team has a defined
Leader who coordinates specific tasks and secondary leaders that have responsibility
For subtasks. Problem solving remains a group activity, but implementation
Of solutions is partitioned among subgroups by the team leader.
Communication among subgroups and individuals is horizontal. Vertical communication
Along the control hierarchy also occurs.

Controlled Centralized (CC). Top-level problem solving and internal team
Coordination are managed by a team leader. Communication between the leaders

And team members are vertical.

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

(i) Software Scope

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

Context. How does the software to be built fit into a larger system, product, or business context and what constraints are imposed as a result of the context?

Information objectives. What customer-visible data objects are produced as output from the software? What data objects are required for input?

Function and performance. What function does the software perform to transform input data into output? Are any special performance characteristics to be addressed?

Software project scope must be unambiguous and understandable at the management and technical levels.

3. Process-The generic phases that characterize the software process—definition, development, and support—are applicable to all software. The problem is to select the process model that is appropriate for the software to be engineered by a project team.

- The linear sequential model
- The prototyping model
- The RAD model
- The incremental model
- The spiral model
- The WINWIN spiral model
- The component-based development model
- The concurrent development model
- The formal methods model
- The fourth generation techniques model

4. Project-We conduct planned and controlled software projects for one primary reason—it is the only known way to manage complexity. In order to avoid project failure, a software project manager and the software engineers who build the product must avoid a set of common warning signs, understand the critical success factors that lead to good project management, and develop a commonsense approach for planning, monitoring and controlling the project.

Q.2 Explain the Role of Metrics and measurement?

Ans. Software metrics refers to a broad range of measurements for computer software.

Measurement can be applied to the software process with the intent of improving it on a continuous basis. Measurement can be used throughout a software project to assist in estimation, quality control, productivity assessment, and project control. Finally, measurement can be used by software engineers to help assess the quality of technical work products and to assist in tactical decision making as a project proceeds.

Software process and product metrics are quantitative measures that enable software people to gain insight into the efficacy of the software process and the projects that are conducted using the process as a framework. Basic quality and productivity data are collected. These data are then analyzed, compared against past averages, and assessed to determine whether quality and productivity improvements have occurred.

Metrics are also used to pinpoint problem areas so that remedies can be developed and the software process can be improved. Software metrics are analyzed and assessed by software managers. Measures are often collected by software engineers.

MEASURES, METRICS, AND INDICATORS

Although the terms measure, measurement, and metrics are often used interchangeably, it is important to note the subtle differences between them. Because measure can be used either as a noun or a verb, definitions of the term can become confusing.

Within the software engineering context, a measure provides a quantitative indication of the extent, amount, dimension, capacity, or size of some attribute of a product or process.

METRICS IN THE PROCESS AND PROJECT DOMAINS

Measurement is commonplace in the engineering world. We measure power consumption, weight, physical dimensions, temperature, voltage, signal-to-noise ratio the list is almost endless. Unfortunately, measurement is far less common in the software engineering world. We have trouble agreeing on what to measure and trouble evaluating measures that are collected.

Q 3. What is the Project Metrics?

Ans. Software process metrics are used for strategic purposes. Software project measures are tactical. That is, project metrics and the indicators derived from them are used by a project manager and a software team to adapt project work flow and technical activities.

The first application of project metrics on most software projects occurs during estimation. Metrics collected from past projects are used as a basis from which effort and time estimates are made for current software work. As a project proceeds, measures of effort and calendar time expended are compared to original estimates (and the project schedule). The project manager uses these data to monitor and control progress.

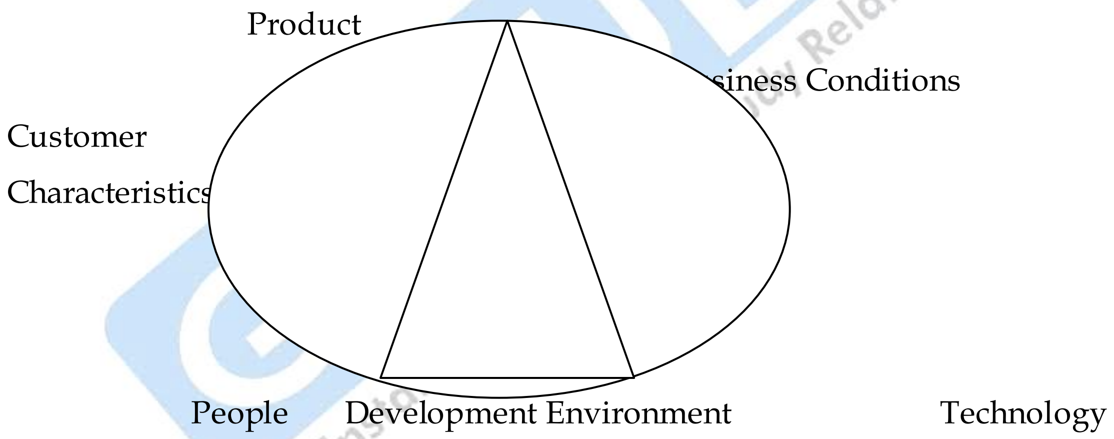
As technical work commences, other project metrics begin to have significance.

Production rates represented in terms of pages of documentation, review hours, function points, and delivered source lines are measured.

Q 4. Explain the Process Metrics?

Ans. Process metrics is used to measure the overall progress of project. It also leads software process improvements.

(1) Some specific attributes of process are measured.
(2) These attributes will develop various metrics.
(3) After executing metrics, indicators will be provided.
(4) Indicators with improve the process.



Determinants for software quality and organizational effectiveness

Process sits at center connecting 3 factors that have profound influence on software quality and organizational performance i.e. People, product and technology.

Process is also related with customer characteristics, Business conditions and development environment.

Customer should have a proper communication, discussion with development team and should be able to present all requirements. Business conditions are checked for how many and what features will be provided by product and what are the limitations of product. All these factors should be directly concerned with the process of the project.

We measure the efficacy of a software process indirectly. That is, we derive a set of metrics based on the outcomes that can be derived from the process. Outcomes include measures of errors uncovered before release of the software, defects delivered to and reported by end-users, work products delivered (productivity), human effort expended, calendar time expended, schedule conformance, and other measures.

We also derive process metrics by measuring the characteristics of specific software engineering tasks.

Q.5 Define in short what is Projection?

Ans.: Projection is a method used in the analysis part of Software Engineering. In Projection, a system is defined from multiple point of view. While using the concept of projection, different view points are defined. Then the system is analysed from these different perspectives.

The advantage of using projection is that the analysis becomes easier. The scope of analysis is limited & it becomes more focussed.

Q.4 What is Software Requirement?

Ans.: In software engineering, requirements analysis is used for determining the needs or conditions to meet for a new or altered product, taking account of the possibly conflicting requirements of the various stakeholders or users. Systematic requirements analysis is also known as requirements engineering. It is sometimes referred loosely by names such as requirements gathering, requirements capture, or requirements specification. Requirements analysis is critical to the success of a development project.

Requirements must be actionable, measurable, testable, related to identified business needs or opportunities, and defined to a level of detail sufficient for system design.

Conceptually, requirements analysis includes three types of activity:

- **Eliciting Requirements :** the task of communicating with customers and users to determine what their requirements are. This is sometimes also called requirements gathering.

- **Analyzing Requirements:** determining whether the stated requirements are unclear, incomplete, ambiguous, or contradictory, and then resolving these issues.
- **Recording Requirements:** Requirements may be documented in various forms, such as natural-language documents, use cases, user stories, or process specifications.

Q.5 Define Partitioning & explain its different types.

OR

What is Horizontal & Vertical Partitioning?

Ans.: The basic principle used in analysis uses the same rule as "divide & conquer" i.e. partition the problem into subproblems & then each subproblem is much easier to understand. With this, the relationship of these subproblems with other subproblems helps to understand the whole problem.

Partitioning decomposes a problem into its consitutent parts. We normally a hierarchial representation of function are information.

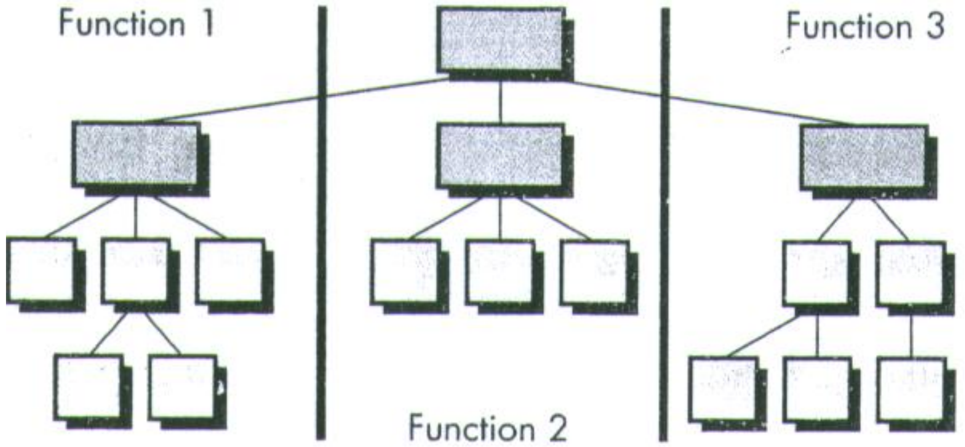
Partitioning could be both vertical as well as horizontal.

Horizontal Partitioning:

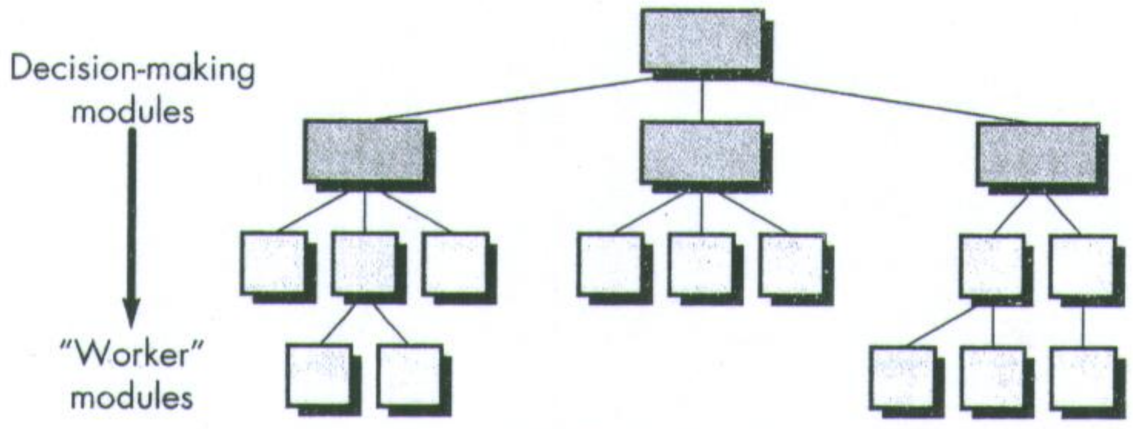
- Horizontal partitioning defines separate branches of the modular hierarchy.
- Control modules are represented in darker shades.
- Here we do three partitions –
  (a) Input
  (b) Data Transformation (Processing)
  (c) Output
- By the decoupling of major functions changes become less complex.

Vertical Partitioning:

- It is often called Factoring.
- Here control & work is distributed top-down.
- The modules which are at lower position, should always perform all inputs, computation & output tasks.



(a) Horizontal partitioning



(b) Vertical partitioning

Q.6 What is Software Requirements Specification (SRS)?

Ans.: A **Software Requirements Specification** (**SRS**) is a complete description of the behaviour of the system to be developed. It includes a set of use cases that describe all the interactions that the users will have with the software. Use cases are also known as "functional requirements". In addition to use cases, the SRS also contains nonfunctional (or supplementary) requirements. "Non-functional requirements" are requirements which impose constraints on the design or implementation (such as performance engineering requirements, quality standards, or design constraints).

Q.7 Explain Information Flow Model.

Ans.: Informaton is transformed as it flows through a computer based system.

A system takes input in many forms, processes it & creates a desired output. But the transformation could be a single logical statement or a complex algorithm.

To understand such working we can create a flow model. With the help of this, we can understand a complete information transformation process.

- o Here a rectangle is used to represent an external entity.
- o A circle represents a process.
- o An arrow represents data objects.
- o A double line shows a data store.

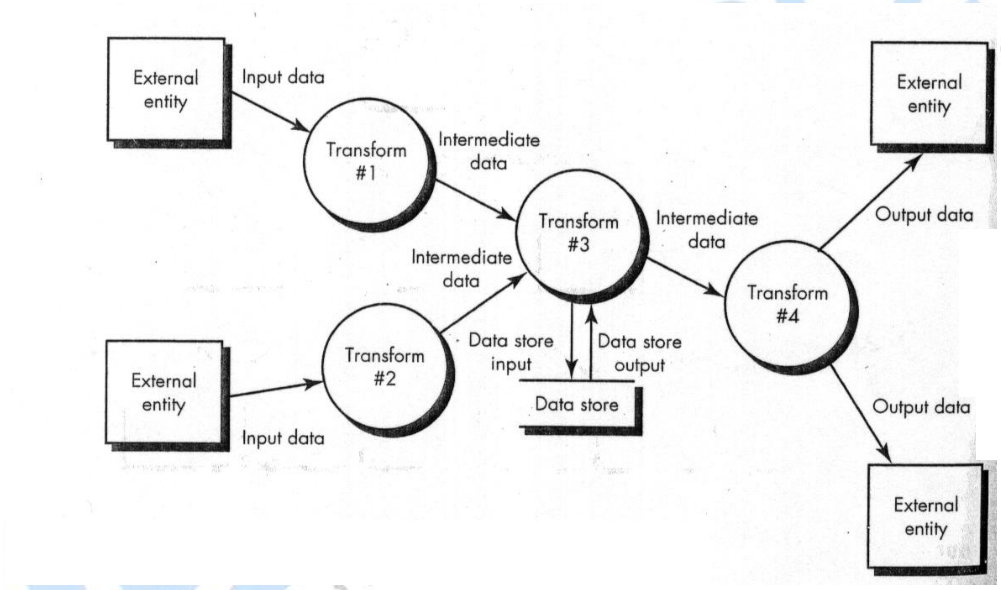


Diagram: Information Flow Model

Q.8 Explain in brief Data Model Analysis.

Ans.: It is useful for any data processing application. It considers data independent of the processing.

The Data Model consists of three interrelated pieces of information –

i) Data Object

ii) Attribute

iii) Relationship

Data Object: Data Object is a representation of almost any composite information. It could be any external entity which could be a thing like reporter test, an occurance like alarm or a unit like accounting department. Data objects encapsulate data only. But they are related like a person can own a car where person & car are objects.

Attribute: It defines the properties of data object. They can be used to name an instance of the data object, describes the instance or make reference to other instance.

Relationship: It consists of no. of attributes. The Data ojects are connected to one another in different ways.

Q.9 Explain in short the Object Oriented Analysis.

Ans. **Object-Oriented Analysis and Design** (OOAD) is a software engineering approach that models a system as a group of interacting objects. Each object represents some entity of interest in the system being modeled, and is characterised by its class, its state (data elements), and its behavior. Various models can be created to show the static structure, dynamic behavior, and run-time deployment of these collaborating objects. Object-oriented analysis (OOA) applies object-modeling techniques to analyze the functional requirements for a system. Object-oriented design (OOD) elaborates the analysis models to produce implementation specifications. OOA focuses on what the system does, OOD on how the system does it.

Object-oriented analysis (OOA) looks at the problem domain, with the aim of producing a conceptual model of the information that exists in the area being analyzed. Analysis models do not consider any implementation constraints that might exist, such as concurrency, distribution, persistence, or how the system is to be built. Implementation constraints are dealt with during object-oriented design (OOD). Analysis is done before the Design.

The sources for an analysis can be a written requirements statement, a formal vision document, and interviews with stakeholders or other interested parties. A system may be divided into multiple domains, representing different business, technological, or other areas of interest, each of which are analyzed separately.

The result of object-oriented analysis is a description of what the system is functionally required to do, in the form of a conceptual model.

Object-oriented design (OOD) transforms the conceptual model produced in object-oriented analysis to take account of the constraints imposed by the chosen architecture and any non-functional – technological or environmental

– constraints, such as transaction throughput, response time, run-time platform, development environment, or programming language.

The concepts in the analysis model are mapped onto implementation classes and interfaces. The result is a model of the solution domain, a detailed description of how the system is to be built.

□ □ □

Chapter-3

Software Project Management

Q.1 Explain the characterstics of a software product.

Ans.: The software product should have following characteristics and sub-characteristics

- **Functionality** - *A set of attributes that bear on the existence of a set of functions and their specified properties. The functions are those that satisfy stated or implied needs.*
  - Suitability
  - Accuracy
  - Interoperability
  - Compliance
  - Security
- **Reliability** - *A set of attributes that bear on the capability of software to maintain its level of performance under stated conditions for a stated period of time.*
  - Maturity
  - Recoverability
  - Fault Tolerance
- **Usability** - *A set of attributes that bear on the effort needed for use, and on the individual assessment of such use, by a stated or implied set of users.*
  - Learnability
  - Understandability
  - Operability
- **Efficiency** - *A set of attributes that bear on the relationship between the level of performance of the software and the amount of resources used, under stated conditions.*
  - Time Behaviour

- Resource Behaviour

- **Maintainability** - *A set of attributes that bear on the effort needed to make specified modifications.*
  - Stability
  - Capacity to Analyse
  - Changeability
  - Testability
- **Portability** - *A set of attributes that bear on the ability of software to be transferred from one environment to another.*
  - Installability
  - Replaceability
  - Adaptability

Q.2 Define Project Tables.

Ans.: After the creation of Time Line Chart as an input many planners produce a different scheduling tool known as project table.

It is a tabular listing of all project tasks which include their planned & actual start to end dates & different related information.

These tools actually enable the project manager to track progress.

Here the work task is placed on the left column of table after this the planning dates & then the actual dates.

Q.3 Define Time Box.

Ans.: Project scheduling provides a road map for a project manager. But when the difference between planned time & actual time increases, the managers face dead line pressures. To overcome this sometimes they use time boxing technique.

It is a scheduling & controlling technique. It recognizes that the complete product may not be deliverable on the predefined deadline. For this an incremental software paradigm is selected & a schedule is then derived for each incremental delivery.

The tasks associated with each increment are time boxed, which means the schedule for each task is adjusted by working backward from the delivery date. A box is put around each task, when a task hits the boundary of its time box, work stops & new task begins.

Q 4. Define the Decomposition Technique?

Ans. It takes a divide and conquers approaches to software project estimation. By decomposing a project into major functions and related software engineering activities, cost and effort estimation can be performed.

The decomposition approach was discussed from two different points of view: decomposition of the problem and decomposition of the process. Estimation uses one or both forms of partitioning. But before an estimate can be made, the project planner must understand the scope of the software to be built and generate an estimate of its "size."

1) Software Sizing

The accuracy of a software project estimate is predicated on a number of things: (1) the degree to which the planner has properly estimated the size of the product to be built; (2) the ability to translate the size estimate into human effort, calendar time, and dollars (a function of the availability of reliable software metrics from past projects); (3) the degree to which the project plan reflects the abilities of the software team; and (4) the stability of product requirements and the environment that supports the software engineering effort.

Putnam and Myers suggest four different approaches to the sizing problem:

"Fuzzy logic" sizing. This approach uses the approximate reasoning techniques that are the cornerstone of fuzzy logic. To apply this approach, the planner must identify the type of application, establish its magnitude on a qualitative scale, and then refine the magnitude within the original range.

Although personal experience can be used, the planner should also have access to a historical database of projects8 so that estimates can be compared to actual experience.

Function point sizing. The planner develops estimates of the information domain characteristics discussed in Chapter 4.

Standard component sizing. Software is composed of a number of different "standard components" that are generic to a particular application area.

For example, the standard components for an information system are subsystems, modules, screens, reports, interactive programs, batch programs, files, LOC, and object-level instructions. The project planner estimates the number of occurrences of each standard component and then uses historical project data to determine the delivered size per standard component. To illustrate, consider an information systems application. The planner estimates

that 18 reports will be generated. Historical data indicates that 967 lines of COBOL
[PUT92] are required per report. This enables the planner to estimate that 17,000 LOC will be required for the reports component. Similar estimates and computation are made for other standard components, and a combined size value (adjusted statistically) results.
Change sizing. This approach is used when a project encompasses the use of existing software that must be modified in some way as part of a project.
The planner estimates the number and type (e.g., reuse, adding code, changing code, and deleting code) of modifications that must be accomplished. Using an "effort ratio" for each type of change, the size of the change may be estimated.

2) Problem-Based Estimation

Lines of code and function points were described as measures from which productivity metrics can be computed. LOC and FP data are used in two ways during software project estimation: (1) as an estimation variable to "size" each element of the software and (2) as baseline metrics collected from past projects and used in conjunction with estimation variables to develop cost and effort projections.
LOC and FP estimation are distinct estimation techniques. Yet both have a number of characteristics in common. The project planner begins with a bounded statement of software scope and from this statement attempts to decompose software into problem functions that can each be estimated individually. LOC or FP (the estimation variable) is then estimated for each function. Alternatively, the planner may choose another component for sizing such as classes or objects, changes, or business processes affected.
Baseline productivity metrics (e.g., LOC/pm or FP/pm9) are then applied to the appropriate estimation variable, and cost or effort for the function is derived. Function estimates are combined to produce an overall estimate for the entire project.
It is important to note, however, that there is often substantial scatter in productivity metrics for an organization, making the use of a single baseline productivity metric suspect. In general, LOC/pm or FP/pm averages should be computed by project domain. That is, projects should be grouped by team size, application area, complexity, and other relevant parameters. Local domain averages should then be computed. When a new project is estimated, it should first be allocated to a domain, and then the appropriate domain average for productivity should be used in generating the estimate.
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, decomposition10 is absolutely essential and is often taken to considerable levels of detail.

3) An Example of LOC-Based Estimation

As an example of LOC and FP problem-based estimation techniques, let us consider a software package to be developed for a computer-aided design application for mechanical components. A review of the *System Specification* indicates that the software is to execute on an engineering workstation and must interface with various computer graphics peripherals including a mouse, digitizer, high resolution color display and laser printer.

Using the *System Specification* as a guide, a preliminary statement of software scope can be developed:

The CAD software will accept two- and three-dimensional geometric data from an engineer. The engineer will interact and control the CAD system through a user interface that will exhibit characteristics of good human/machine interface design. All geometric data and other supporting information will be maintained in a CAD database. Design analysis modules will be developed to produce the required output, which will be displayed on a variety of graphics devices. The software will be designed to control and interact with peripheral devices that include a mouse, digitizer, laser printer, and plotter.

4) Process-Based Estimation

The most common technique for estimating a project is to base the estimate on the process that will be used. That is, the process is decomposed into a relatively small set of tasks and the effort required to accomplish each task is estimated.

Like the problem-based techniques, process-based estimation begins with delineation of software functions obtained from the project scope. A series of software process activities must be performed for each function.

Once problem functions and process activities are melded, the planner estimates the effort that will be required to accomplish each software process activity for each software function. Average labor rates (i.e., cost/unit effort) are then applied to the effort estimated for each process activity. It is very likely the labor rate will vary for each task. Senior staff heavily involved in early activities is generally more expensive than junior staff involved in later design tasks, code generation, and early testing.

Q.5 Explain Software Sizing & Size Matrix.

Ans.: The **Software Size** is the most important factor that affects the software cost. This section describes five software size metrics used in practice. The lines of code and function points are the most popular matrics among the five metrices.

Line of Code : This is the number of lines of the delivered source code of the software, excluding comments and blank lines and is commonly known as LOC. Although LOC is programming language dependent, it is the most widely used software size metric. Most models relate this measurement to the software cost. However, exact LOC can only be obtained after the project has completed. Estimating the code size of a program before it is actually built is almost as hard as estimating the cost of the program. A typical method for estimating the code size is to use expert's judgment together with a technique called PERT. It involves expert's judgment of three possible code-sizes : S_1, the lowest possible size; S_h, the highest possible size; and S_m, the most likely size. The estimate of the code-size S is computed as :

$$S = \frac{S_1 + S_h + 4S_m}{6}$$

PERT can also be used for individual components to obtain an estimate of the software system by summing up the estimates of all the components.

Function Points : This is a measurement based on the functionality of the program and was first introduced by Albrecht. The total number of function points depends on the counts of distinct (in terms of format or processing logic) types in the following five classes :-

(i) **User-Input Types :** Data or Control User Input Types.

(ii) **User-Output Types :** Output Data Types to the user that leaves the system.

(iii) **Inquiry Types :** Interactive inputs requiring a response.

(iv) **Internal File Types :** Files (Logical Group of Information) that are used and shared inside the System.

(v) **External File Types :** Files that are passed or shared between the System and other Systems.

Each of these types is individually assigned one of three complexity levels of {1 = simple, 2 = medium, 3 = complex) and given a weighting value that varies from 3 (for simple input) to 15 (for complex internal files).

The Unadjusted Function-point Count (UFC) is given as :

$$UFC = \sum_{i=1}^{5} \sum_{j=1}^{3} N_6 W_6$$

Where N_6 and W_6 are respectively the number and weight of types of class i with complexity j.

For example if the raw function-point counts of a project are 2 simple inputs (W_6=3), 2 complex outputs (W_6=7) and 1 complex file (W_6=15). Then UFC = 2*3 + 2*7 + 1*15 = 35.

The initial function-point count is either directly used for cost estimation or is further modified by factors whose values depend on the overall complexity of the project. This will take into account the degree of distributed processing, the amount of reuse, the performance requirement, etc. The final function-point count is the product of the UFC and the project complexity factors. The advantage of the function-point measurement is that it can be obtained based on the system requirement specification in the early stage of software development. The UFC is also used for code-size estimation using the following linear formula :

$$LOC = a * UFC + b$$

The parameters a, b can be obtained using linear regression and previously completed project data.

Q.6 Explain Cost Estimation.

Ans.: There are two major types of cost estimation methods : Algorithmic and Non-Algorithmic. Algorithmic Models vary widely in mathematical sophistication. Some are based on simple arithmetic formulas using such summary statistics as means and standard deviations. Others are based on regression models and differential equations. To improve the accuracy of algorithmic models, there is a need to adjust or calibrate the model to local circumstances. These models cannot be used off-the-shelf. Even with calibration the accuracy can be quite mixed.

Q.7 Describe the Cost Factors that affects the Estimation.

Ans.: Besides the Software Size, there are many other Cost Factors. The most comprehensive set of Cost Factors are proposed and used by Boehm in the COCOMO II Model. These Cost Factors can be divided into four types :-

(i) **Product Factors:** Required reliability; product complexity; database size used; required reusability; documentation match to life-cycle needs.

(ii) **Computer Factors:** Execution time constraint; main storage constraint; computer turnaround constraints; platform volatility.

(iii) **Personal Factors:** Analyst capability; application experience; programming capability; platform experience; language and tool experience; personnel continuity.

(iv) **Project Factors:** Multisite development; use of software tool; required development schedule.

Q.8 Define COCOMO (Constructive Cost Model) in detail.

Ans.: This family of models was proposed by Boehm. The models have been widely accepted in practice. In the COCOMOs, the code-size S is given in thousand LOC (KLOC) and efforts is on person-month.

COCOMO is actually a hierarchy of estimation models that address 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.

(A) **Basic COCOMO :** This model uses three sets of {a, b} depending on the complexity of the software only :-

(i) For simple, well-understand applications, a = 2.4, b = 1.05;

(ii) For more complex systems, a = 3.0, b = 1.15;

(iii) For embedded systems, a = 3.6, b = 1.20.

The basic COCOMO model is simple and easy to use. As many Cost Factors are not considered, it can only be used as a rough estimate.

(B) **Intermediate COCOMO and Detailed COCOMO :** In the intermediate COCOMO, a nominal effort estimation is obtained using the power function with three sets of {a, b}, with coefficient a being slightly different from that of the basic COCOMO :-

(i) For simple, well-understood applications, a = 3.2, b = 1.05;

(ii) For more complex systems, a = 3.0, b = 1.15;

(iii) For embedded systems, a = 2.8, b = 1.20.

Then, fifteen Cost Factors with values ranging from 0.7 to 1.66 (See Table) are determined. The overall impact factor M is obtained as the product of all individual factors, and the estimate is obtained by multiplying M to the nominal estimate.

Cost Factors and their weights in COCOMO II

Cost Factors	**Description**	**Rating**				
		Very Low	**Low**	**Nominal**	**High**	**Very High**
	Product					
RELY	Required Software Reliability	0.75	0.88	1.00	1.15	1.40
DATA	Database Size	-	0.94	1.00	1.08	1.16
CPLX	Product Complexity	0.70	0.85	1.00	1.15	1.30
	Computer					
TIME	Execution Time Constraint	-	-	1.00	1.11	1.30
STOR	Main Storage Constraint	-	-	1.00	1.06	1.20
VIRT	Virtual Machine Volatility	-	0.87	1.00	1.15	1.30
TURN	Computer Turnaround Time	-	0.87	1.00	1.07	1.15
	Personal					
ACAP	Analyst Capability	1.46	1.19	1.00	0.86	0.71
AEXP	Application Experience	1.29	1.13	1.00	0.91	0.82
PCAP	Programmer Capability	1.42	1.17	1.00	0.86	0.70
VEXP	Virtual Machine Experience	1.21	1.10	1.00	0.90	-
LEXP	Language Experience	1.14	1.07	1.00	0.95	-
	Project					

MODP	Modern Programming Practice	1.24	1.10	1.00	0.91	0.82
TOOL	Software Tools	1.24	1.10	1.00	0.91	0.83
SCED	Development Schedule	1.23	1.08	1.00	1.04	1.10

While both basic and intermediate COCOMOs estimate the software cost at the system level, the detailed COCOMO works on each sub-system separately and has an obvious advantage for large systems that contain non-homogenous subsystems.

Q.9 Describe the different tools used for scheduling.

OR

Define Time Line Chart.

OR

What is Gantt chart.

Ans.: Time Line Chart:

When a project is scheduled the planner begins with a set of tasks. These inputs are known as "work breakdown imputs". As a result a time line chart is generated.

This chart can be developed for each task or for the entire project. Here all project tasks are listed in the left side column.

The horizontal bars indicate the duration of each task.

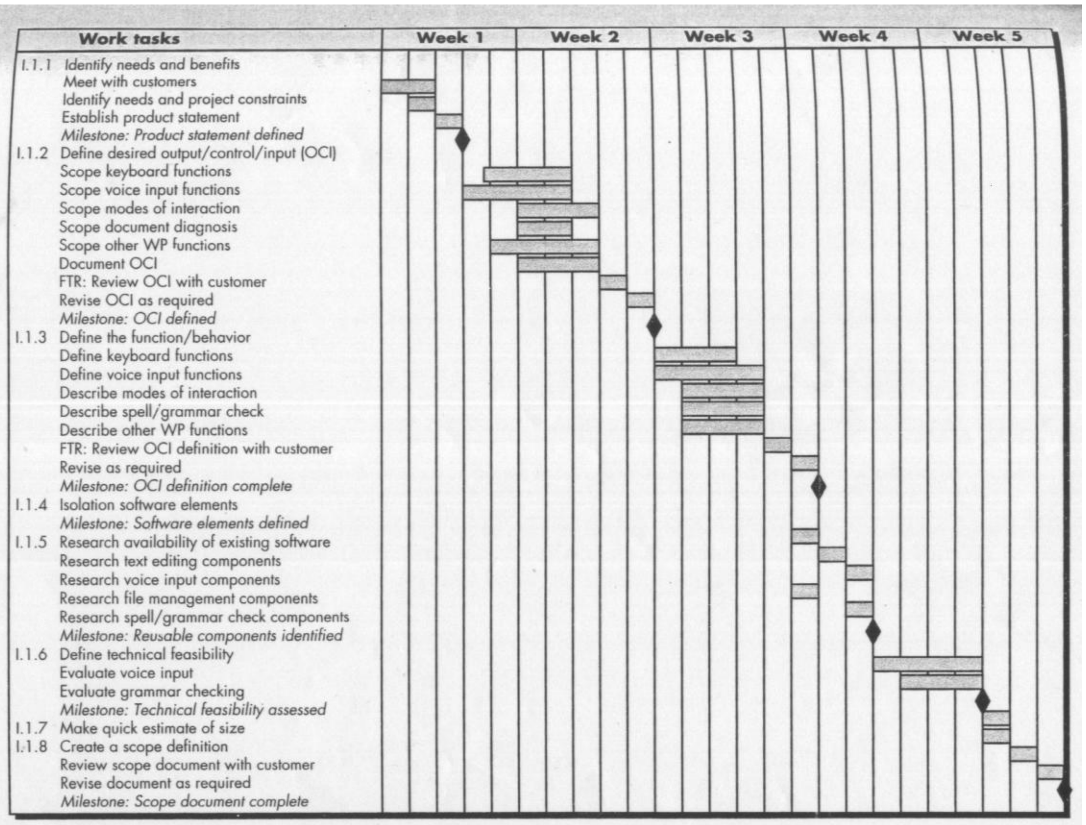


Diagram : Time Line Chart

Q.10 How can a project be monitored?

Ans.: The monitoring of project could be done by using the following points:

1. **Keep it simple**
   - Remember… monitoring is meant to be a help to good project management and not a burden.
2. **Objectives**
   - Work out clearly at the beginning the objectives of the project, including a budget of the likely cost (expenditure).
3. **Plan the activities**
   - what needs to be done
   - when it should be done
   - who will be involved in doing it
   - what resources are needed to do it

- how long it will take to do
- how much it will cost.

4. **Monitoring**

   Work out the most appropriate way of monitoring the work - again, keep it simple:
   - meetings
   - diaries
   - reports on progress
   - accounts, reports on finances.

Monitoring methods :

- **Reports**

  These do not have to be very long. Their purpose needs to be clear - to report on activities and achievements. They must give a clear and helpful record of exactly what has been achieved. Reports should be short and to the point.

  The ideal report - like the one written below - is short and to the point.

Report of Courses Conducted

Objectives	Outcome	Evaluation
Consciousness raising 1 Conduct 18 courses. (average size 18)	19 courses held : 10 for men, 9 for women. Total participants : 332 Average size : 17.5 Course Length : 3 days	These courses are very effective in motivating group members. Groups have been transformed when members have received this training.
Objectives	**Outcome**	**Evaluation**
Group management Design module. Conduct 11 courses. (average size 18)	Module designed. 5 courses held. Total participants : 63 Average size : 12.6 Course length : 3 days	Purpose of course was to provide training in keeping group records. Because so many members are illiterate, there were not sufficient members to join these courses. Instead, we are now

		teaching the subject at each group meeting.
Conduct 5 Child Literacy courses	This program has been postponed.	We decided that this program should be done when whole villages have been mobilized. We are not at this stage yet.

- **Diaries**

  A helpful way of recording information is to use one side of a note book for example, for daily or weekly plans. Write on the other side what actually happened. Keeping a work diary like this will help you also to evaluate your own work. What are you doing that is most helpful and bring effective results? Take time to ask people in the community about how they feel.

- **Finances**

  Donor agencies often transfer funds in quarterly or half yearly payments which may not fit in with the planned project expenses. Planning of expenditure may need to take this into account. Careful budgeting and planning will be of great help in this.

- **Meetings**

  Confidence and trust are vital. There is a possibility that monitoring may be seen as a way of checking up on people. It must be a joint effort in which everyone is involved in some way or another.

Q.11 Define Project Control.

Ans.: During the execution of a project, procedures for project control become indispensable tools to managers and other participants in the construction process. These tools serve the dual purpose of recording is that occur as well as giving managers an indication of the progress and problems associated with a project. The problems of project control are aptly summed up in an old definition of a project as "any collection of vaguely related activities that are ninety percent complete, over budget and late." The task of project control systems is to give a fair indication of the existence and the extent of such problems.

The limited objective of project control deserves emphasis. Project control procedures are primarily intended to identify deviations from the project plan rather than to suggest possible areas for cost savings. This characteristic reflects the advanced stage at which project control becomes important. The time at which major cost savings can be achieved is during planning and design for the project. During the actual construction, changes are likely to delay the project and lead to inordinate cost increases. As a result, the focus of project control is on fulfilling the original design plans or indicating deviations from these plans, rather than on searching for significant improvements and cost savings. It is only when a rescue operation is required that major changes will normally occur in the construction plan.

Q.12 Write a short note on cost control.

Ans.: For cost control on a project, the construction plan and the associated cash flow estimates can provide the baseline reference for subsequent project monitoring and control. For schedules, progress on individual activities and the achievement of milestone completions can be compared with the project schedule to monitor the progress of activities. Contract and job specifications provide the criteria to assess and assure the required quality of construction. The final or detailed cost estimate provides a baseline for the assessment of financial performance during the project. To the extent that costs are within the detailed cost estimate, then the project is thought to be under financial control. Overruns in particular cost categories signal the possibility of problems and give an indication of exactly what problems are being encountered. Expense oriented construction planning and control focuses upon the categories included in the final cost estimation. This focus is particular relevant for projects with few activities and considerable repetition.

For control and monitoring purposes, the original detailed cost estimate is typically converted to a *project budget*, and the project budget is used subsequently as a guide for management. Specific items in the detailed cost estimate become job cost elements. Expenses incurred during the course of a project are recorded in specific job cost accounts to be compared with the original cost estimates in each category. Thus, individual job cost accounts generally represent the basic unit for cost control. Alternatively, job cost accounts may be disaggregated or divided into *work elements* which are related both to particular scheduled activities and to particular cost accounts.

□ □ □

Chapter-4

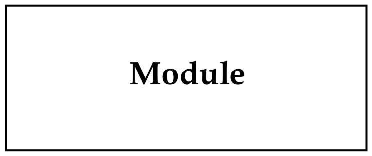
Software Design

Q.1 Define the tool designed by Yordon for structured designing.

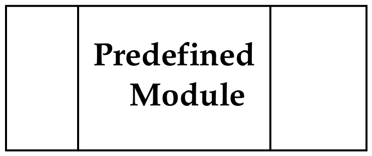
Ans.: Structured design partitions a program into small, independent modules. That is, a system is broken down, decomposed, or otherwise partitioned into well-defined processing "modules". They are arranged in a hierarchy or top-down manner. Structured design is an attempt to minimize complexity and make a problem manageable by dividing it into smaller segments called modularization. The tool used for structured design is the Structure Chart.

Yourdon Defined Structure Chart : A structure chart is used to show the programmers of a system how the system is partitioned into modules and identify the parameters or information passed between modules. It is used more to identify the external functions, inputs and output (of modules rather than their internal procedures and data. A module is simply a well-defined process within the system.

A module is represented by a rectangle with the module's name inside. Modules are typically named by a verb phrase that describes the processing to be performed.



A predefined module is a special type of module that does not need to be implemented because it is part of some external library of modules.



A n arrow from the calling module to the called module denotes module calls.

Module Call

A completed Structure Chart gives an overall appearance of increasing detail form top to bottom. The top module (s) are the 'boss' modules which call or trigger calls to all other subordinate module. There is no specified order of calling from one module to the next, as the Structure Chart is simply a hierarchical representation of the modules in the System.

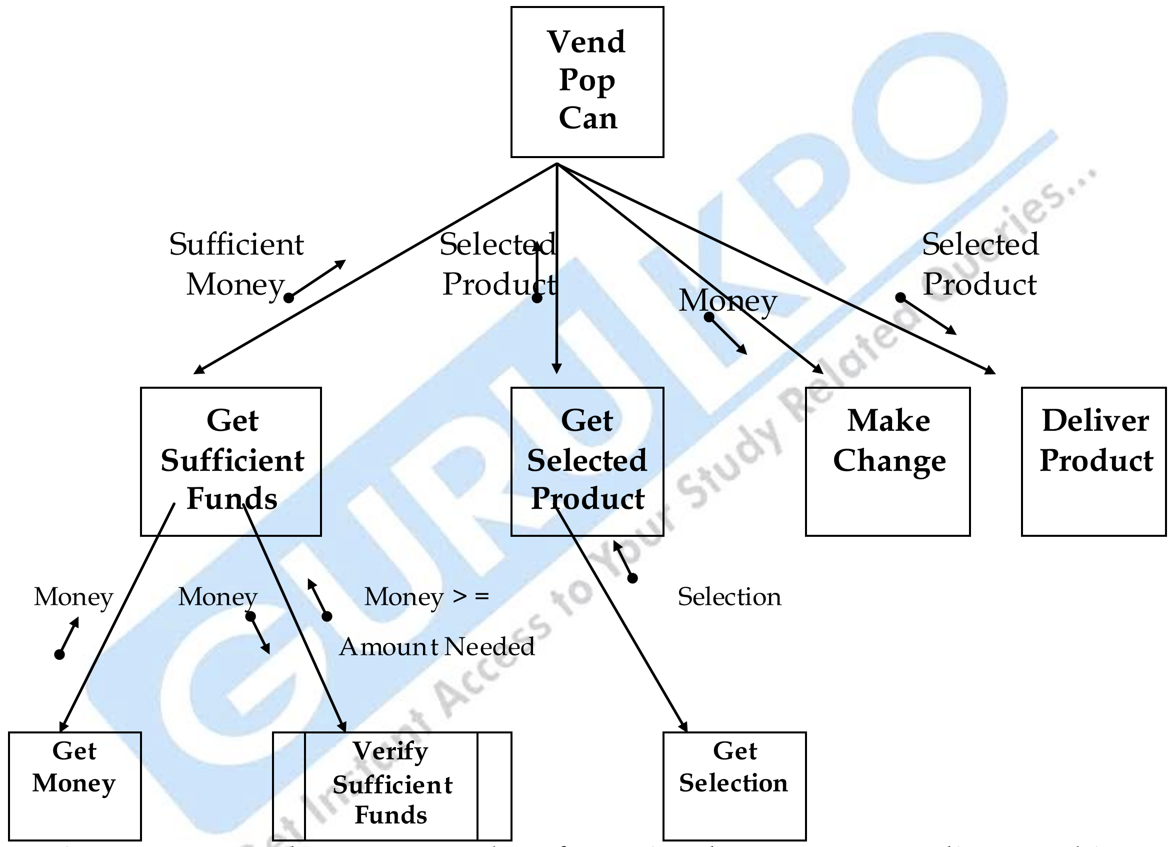


Diagram : Example Structure Chart for a Simple Pop Can Vending Machine

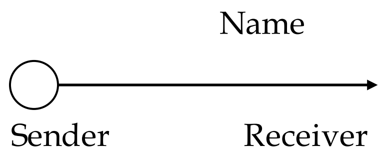
In this figure, the 'boss' module is "Vend Pop Can" with all other modules being called directly or indirectly from this top-level module.

Q.2 Define Data Couples & Flags.

Ans.: Communication between modules are handled by data couples and flags.

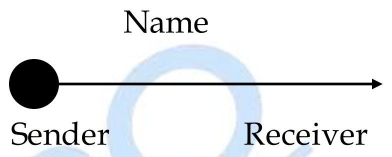
Data Couples : A data couple is simply a piece of data that is passed between modules, and is used by the modules within their processing function. Data couples are typically named using a noun, as they represent a 'physical' piece of data or entity.

Data Couple processed externally relevant :



Flags: Flags are used when state information is needed to be sent to other modules. Figure (2) illustrates the symbols used to denote flags and data couples. Flags fall into two categories - descriptive and control.

Flag set and tested internally relevant



- **Descriptive flags** are used to communicate state information between modules and are usually with an adjective, (eg : valid details).
- **Control Flags** are used to provide a directive from one module to another, and so they are most often named with a verb phase (eg. Money >= amount needed).

Q.3 Define Coupling.

Ans.: Coupling refers to the number of connections between a calling and a called module and the complexity of these connections. There must be at least one connection between a module and the calling module. The main objective is to make the modules as independent as possible. Coupling is a measure of the interrelatedness, or interdependence between modules within a system. A good design generally has low coupling.

Q.4 Define Cohesion.

Ans.: An important measure of design quality is cohesion. It refers to the relationship among elements within a module. If a module does more than one task, the instructions in that module are said to be more cohesive (and less error-prone) than modules that perform multiple tasks. Higher or stronger cohesion usually indicates better partitioning, and so, better system design. A module with low cohesion will have elements (data, instructions, module calls) that have little in common with each other or with the modules overall function. A module with high cohesion is quite the opposite, with most or all elements in some way being directly related to the modules main function.

Q5 Define Design Objective ?

Ans. The various desirable objective of software Design are:

1. **Correctness:** The Design of a System is correct if a system built precisely according to the design satisfies the requirement of that system. The goal during the design phase is to produce correct designs. However, correctness is not the sole criterion during the design phase, as there can be many correct designs.
2. **Simplicity:** Simplicity is perhaps the most the most important quality criteria for software systems. Maintenance of software system is usually quite expensive. The design of system is one of the most important factors affecting the maintainability of the system.
3. **Verifiability:** Design should be correct and it should be verified for correctness. Verifiability is concerned with how easily the correctness of the design can be checked. Various verification techniques should be easily applied to design.
4. **Traceability:** Traceability is an important property that can get design Verification. it requires that the entire design element must be traceable to the requirements.
5. **Efficiency:** Efficiency of any system is concerned with the proper use of scarce resources by the system. The need for efficiency arises due to cost considerations. If some resources are scarce and expensive, it is desirable that those resources be used efficiently.

In Computer systems the resource that are most often considered for efficiently are processor time and memory. Two of the important such resources are processor time and memory. An efficient system consumes less processor time and memory.

Q 6 Define Design Principles?

Ans. The goal of design is to satisfy the requirements of the customer. So, there should be some design principles which lead the software engineering to achieve a good design product.

1. The design process should not suffer from "tunnel vision". A good designer should incorporate various alternative approaches.
2. Design should not reinvent the wheel. As time is short and resources are limited so design time should be invested in representing truly new ideas, not to check the previously generated components.

3. The design should follow the same idea of the problem so that the developed product will be according to needs of the problem.
4. The design should be structured to accommodate change. The design should be structured so as to be amenable to change and also maintenance should be easily and properly done.
5. Design is not coding, coding is not design. Each when detailed procedural designs are created for program components, the level of abstraction of design model is higher than code.
6. The Design should be assessed for quality as it is being created, not after the fact. During design phase it is needed that we should consider the quality concern regularly at each step.
7. The design should be reviewed to minimize conceptual errors. To minimize errors, design should be reviewed formally and proper considerations should be given on ambiguity, inconsistency etc.

Q 7 Define Design Concepts?

Ans . A set of fundamental software design concepts have evolved over the past four decades. Although the degree of interest in each concept has varied over the years each has stood the test of time. Each provides the software designer with a foundation from which more sophisticated design methods can be applied. Some of the software design fundamentals are:

1. Abstraction
2. Refinements
3. Modularity
4. Software Architecture
5. Control Hierarchy
6. Structural portioning
7. Data Structure
8. Software procedure
9. Information Hiding

1. **Abstraction** - Abstraction is the process or result of generalization by reducing the information content of a concept or an observable phenomenon,

typically in order to retain only information which is relevant for a particular purpose.

2. **Refinement** - It is the process of elaboration. A hierarchy is developed by decomposing a macroscopic statement of function in a stepwise fashion until programming language statements are reached. In each step, one or several instructions of a given program are decomposed into more detailed instructions. Abstraction and Refinement are complementary concepts.
3. **Modularity** - Software architecture is divided into components called modules.
4. **Software Architecture** - It refers to the overall structure of the software and the ways in which that structure provides conceptual integrity for a system. A good software architecture will yield a good return on investment with respect to the desired outcome of the project, e.g. in terms of performance, quality, schedule and cost.
5. **Control Hierarchy** - A program structure that represents the organization of a program component and implies a hierarchy of control.
6. **Structural Partitioning** - The program structure can be divided both horizontally and vertically. Horizontal partitions define separate branches of modular hierarchy for each major program function. Vertical partitioning suggests that control and work should be distributed top down in the program structure.
7. **Data Structure** - It is a representation of the logical relationship among individual elements of data.
8. **Software Procedure** - It focuses on the processing of each modules individually
9. **Information Hiding** - 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

Q.8 Define Abstraction.

Ans.: Abstraction is a very powerful concept used in Software Engineering. It is a tool that helps the designer to consider a component at an abstract level i.e. without considering the details of the implementation of the component.

An abstraction of a component describes the external behavior "without bothering with the internal points that affect the behavior.

Abstraction can be used for existing components as well as components which are being designed. Abstraction of existing component plays an important role in maintenance phase of software. This actually helps in determining how modifying a component can affect the system.

Q.9 List & Describe the different types of Abstraction.

Ans.: Abstractions can be of four types :

i) Data Abstraction

ii) Procedural Abstraction

iii) Control Abstraction

iv) Functional Abstraction

Data Abstraction: It is a named collection of data or we can define it as the combination of attributes.

Procedural Abstraction: Is a named sequence of instruction. This sequence has a specific & limited function.

Control Abstraction : In this abstraction a program control mechanism is created without specifying internal details.

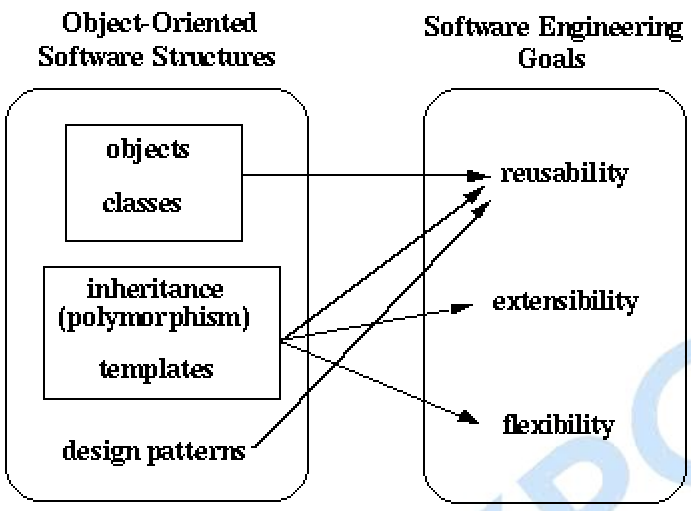
Functional Abstraction : In this type of abstraction a module is specified by the function. It acts as the basis of partitioning.

Q 10. What are the object oriented concepts?

Ans. Object-oriented programming is an evolutionary development in software engineering. The foundation for many object-oriented languages were established by decades of software engineering experience that motivated the invention of language features such as closed procedures, modules and abstract data types. Also important were the widely recognized value of such software engineering techniques as information hiding, encapsulation, strict enforcement of interfaces, and layering.

Object-oriented programming address (at least) the three major software engineering goals shown in this figure:

Software Engineering Goals


Object-Oriented Software Structures
Software Engineering Goals
objects
classes
inheritance (polymorphism)
templates
design patterns
reusability
extensibility
flexibility

The language features that address these issues are those of objects, classes, inheritance, polymorphism, templates, and design patterns.

Reusability is an important issue in software engineering for at least two major reasons. First reusability is one means to cope with the pressures of producing ever larger and more functional systems in an ever decreasing development cycle (time to market). Reusability allows developers to be more efficient because the same code can be developed once and used in many different applications. Second, reliability can be improved by reusing previously developed, and previously tested, components. The development of new code entails the additional costs in time and money of testing, validation, and verification of the new code. Much of these expenses can be avoided by using "off-the-shelf" components.

Software reuse is certainly not a goal unique to object-oriented programming. While libraries of procedures proved this approach to be useful, in practice procedures were too primitive a unit to promote extensive reuse. Objects and classes are more sophisticated mechanisms for achieving software reuse because they bind together more completely all the aspects of an entire abstraction. Therefore, the abstraction can more easily be transported across applications. Any of the forms of generalization also contribute to reuse. A class in an inheritance hierarchy can be reused directly when it serves as a generalized base class from which a new class is derived by specialization. Templates can be reused by supplying different parameters for the template arguments. Design patterns allow design experience and success to be reused across designers.

Q 11. Define Design Methods?

Ans. Software Design follows an ordered sequential steps to consider data and program structure, interface characteristics and procedural details. There are mainly three kinds o design methods.

1) Data design
2) Architecture design
3) Procedural design

1. **Data Design** -Like other software engineering activities, *data design* (sometimes referred to as *data architecting*) creates a model of data and/or information that is represented at a high level of abstraction (the customer/user's view of data). This data model is then refined into progressively more implementation-specific representations that can be processed by the computer-based system. In many software applications, the architecture of the data will have a profound influence on the architecture of the software that must process it.
Data Modeling, Data Structures, Databases, and the Data Warehouse
The data objects defined during software requirements analysis are modeled using entity/relationship diagrams and the data dictionary (Chapter 12). The data design activity translates these elements of the requirements model into data structures at the software component level and, when necessary, a database architecture at the application level.
In years past, data architecture was generally limited to data structures at the program level and databases at the application level. But today, businesses large and small are awash in data. It is not unusual for even a moderately sized business to have dozens of databases serving many applications encompassing hundreds of gigabytes of data. The challenge for a business has been to extract useful information from this data environment, particularly when the information desired is cross functional..

2. Architecture design-

Q.12 What is Documentation? Describe the different manuals used.

Ans.: It is an important part of software. It is a document which could be a paper or electronic document. This works as guideline for all the persons who are connected to the software at any position.

Software normally needs:-

(1) **System Manual:** It provides information about system –

(i) Scope

(ii) Design

(iii) Architecture

(iv) System Flow

(v) Technology Details

(vi) Different Interfaces

The source code is its part, if it is given to customers.

(2) **Operations Manual:** It deals with the system operations. It provides guidelines to users of any use.

(3) **System Maintenance Manual:** It deals with maintenance on day-to-day basis, used by system coordinator. It deals with files & backups.

Q.13 Define Development?

Ans.: The generic phases of development are design, analysis, development, products. The execution of these phases will be executed to develop the software product.

The development process of a software product takes place using any process model. Every software development organization has a setoff activities that are applied to all development projects. They are put into a frame work, called as Common Activity Framework (CAF).

The two-stage and few-stage project life cycle development model is intended to be used for software.

Development is done in increments. This process is interactive & moved through steps like Analysis, Design, Test, Assess, Improve.

Q.14 Describe the different types of case tools used in Software Engineering.

Ans.: **Case Tools :** Case stands for computer aided software engineering. These tools help to reduce the amount of efforts required to produce a product. Case tools are normally used at every step of software process. They actually help the software engineers to automate manual activities. The case tools itself create an environment to the architecture of system. Case tools are not at all tied to project database, but they remain as individual point solutions.

Certain complementary case tools are used to form bridges between the different structures. Like analysis & design tools are used to code generators.

Case tools can be classified by functions or their use in various steps of software development process.

(1) **Process Modeling and Management Tool :** These are used to represent key elements of a process so as to understand them correctly. They also provide links for process descriptions and to other tools to provide support to process activities.

(2) **Project Planning Tools :** These tools focuses on software project effort, cost estimation (labour, hardware, software cost) and process scheduling (Time Chart).

(3) **Project Management Tool :** They help managers to collect matrices for software product qualities.

(4) **Documentation Tools :** Normally, 20-30% of development effort is consumed in documentation to reduce this and to improve the productivity these tools are utilized.

(5) **System Software Tools :** These tools are utilized for high quality network systems softwares and other work station technology.

(6) **Software Configuration Management tools :** These tools can assist 5 major tasks : identification, version control, change control, auditing and status accounting.

(7) **Analysis and Design Tools :** These tools enable a software engineer to create models of the system. They contain behaviour & characteristics of data.

(8) **Programming Tools :** These tools of compilers, editors and debuggers. The object oriented programming environment, database query languages and application generators come under it.

(9) **Integration and Testing Tools :** These testing tools could be further categorized into :

(a) **Data Acquition :** They acquire data to be used during testing.

(b) **Static Measurement :** They analyze source code without executing test cases.

(c) **Dynamic Measurement :** They analyze source code.

(d) **Simulation :** They simulate the functions of many hardware or other external device.

(e) **Test Management :** They consist in planning, development and control of testing.

Q.15 Explain the different types of Design Matrices used in Designing.

Ans.: Design Matrices are utilized while the designing phase occurs in software development. Design Matrices are not perfect but it is always debatable about the manner in which they should be used.

Practically these matrices help the designer to improve insight. The most common among these are :

(1) Architectural Design Matrices

(2) Component Level Matrices

   (a) Cohesion Matrices

   (b) Coupling Matrices

   (c) Complexity Matrices

These matrices focus on characteristics of the program architecture. They can be treated as black box because they do not require any knowledge of inner working of a component. In this depicted three major complexities include are :

- Structural Complexity
- Data Complexity
- System Complexity

   (i) Cohesion

   (ii) Coupling

   (iii) Complexity

Cohesion : 'Viemal and Odd' defined some matrices which calculate the cohesiveness of a module. They define five basic terms for calculating this :

(i) **Data Slice :** It is simply a backward examination at of a module for the data values to effects the location of module.

(ii) **Data Tokens :** They are simply the variables defined for a module.

(iii) **Glue Tokens:** They are the tokens for lies on one or more data slice.

(iv) **Super Glue token:** These tokens are common to every data slice in a module.

(v) **Stickiness :** The relative stickiness of a glue token is directly proportional to number of data slice in binds.

Coupling Metric: They provide the indication of connections of a module to other modules. Thama has proposed a metric for module coupling by calculating data and control flow coupling, global coupling and environmental coupling.

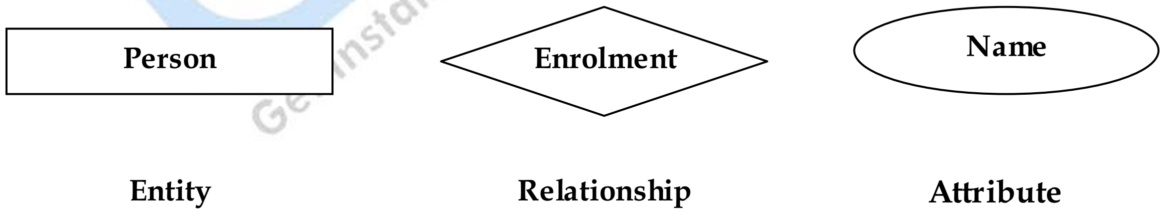
Complexity Metric : A number of software metrics can be computed to determine the complexity of a program in these the most often used is cyclomatic metrics.

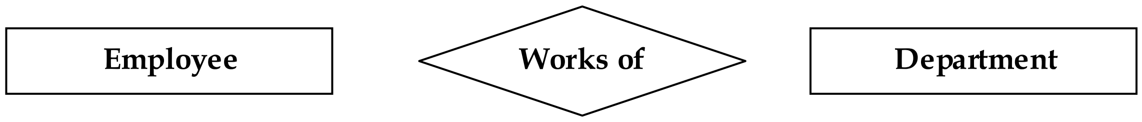
Q.10 Define & explain the Entity - Relationship diagram & its notations.

Ans.: **Entity – Relationship Diagrams :** The object-relationship pair can be represented graphically using an ER diagram. An entity represents an object. Examples : a computer, an employee, a song, a mathematical theorem. Entities are represented as rectangles.

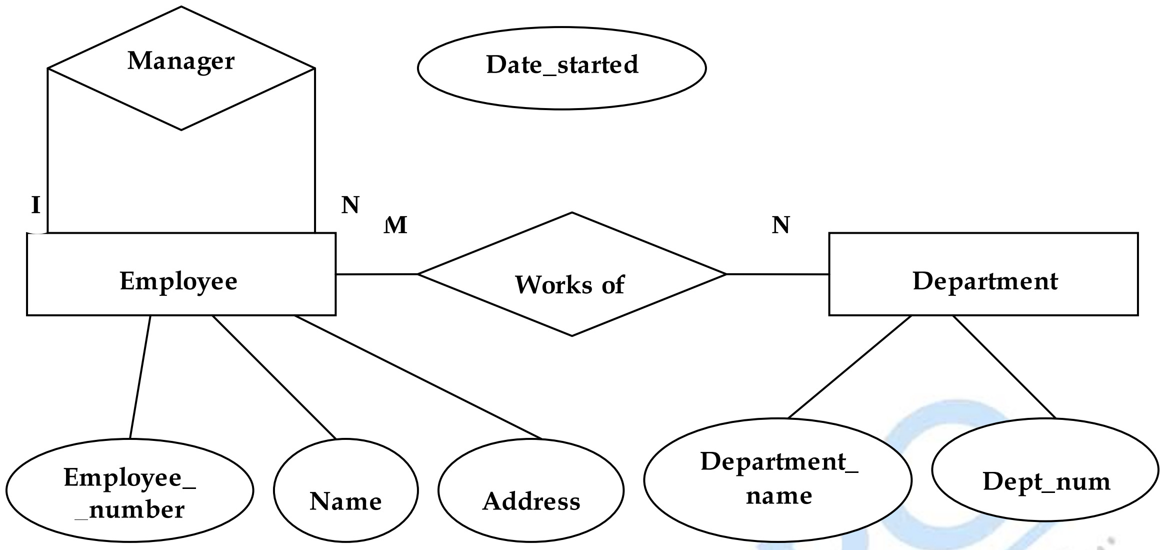
A relationship captures how two or more entities are related to one another. Examples : an relationship between a company and a computer, relationship between an employee and a department, relationship between an artist and a song. Relationships are represented as diamonds, connected by lines to each of the entities.

Entities and relationships can both have attributes. Examples : an employee entity might have an employee ID number attribute, the proved relationship may have a data attribute. Attributes are represented as ellipses connected to their entity by a line.





A simple E-R diagram



An E-R diagram with attributes

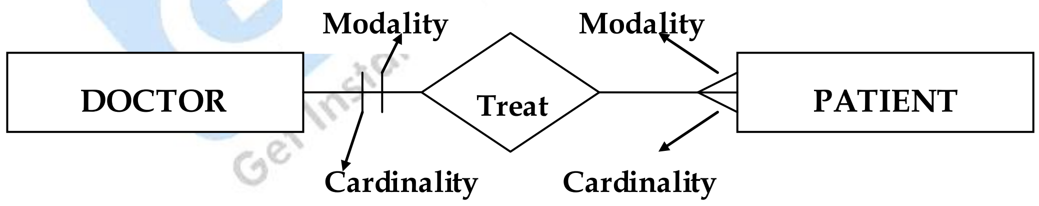
When we need to specify cardinality we use the symbols

- One = a line or dash |
- Many = crow's feet ←

To specify modality we use the symbols

- One = a line or dash |
- Zero = a circle ○

The following ER diagram specifies cardinality and modality.



These symbols on the relationship line that is closest to the data object will denote cardinality and the next will denote modality.

Q.11 Explain the tools developed by Gane & Sarson.

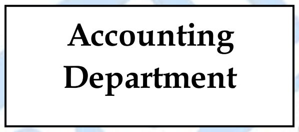
Ans.: Gane & Sarson worked on Structured Analysis and they develop a tool naming Data Flow Diagram.

Data Flow Diagram : A Data Flow Diagram (DFD) is a graphical representation of the "flow" of data through an information system. It describes the system's data and how the process transform the data in a graphical manner. Data flow diagrams can be used to provide a clear representation of any business function.

The result is a series of diagrams that represent the business activities in a way that is clear and easy to communicate. Initially a **context diagram** is drawn, which is a simple representation of the entire system under investigation. This is followed by a level 1 diagram; which provides an overview of the major functional areas of the business. The level 1 diagram identifies the major business processes at a high level and any of these processes can then be analyzed further – giving rise to a corresponding level 2 business process diagram. This process of more detailed analysis can then continue – through level 3, 4 and so on. However, most investigations will stop at level 2 and it is very unusual to go beyond a level 3 diagram.

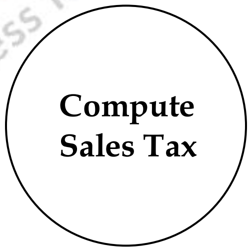
DFD Notation :

A Rectangle :



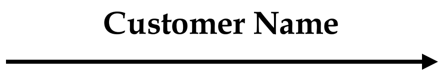
It denotes an external entity. It defines a source of destination of system data. It can represents a person, group of people, department, or some other system.

A Circle :



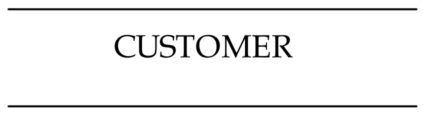
It denotes a process or activity. It is also known as a bubble. It shows how the system transforms inputs into outputs. Each process is named.

A Line with an Arrowhead :



It denotes the direction of data flow. The input to, or output from, a given process, which is associated with each arrow in a DFD.

Open Rectangle :



It is used to model collection of data. It may refer to files or databases, or data stored on punched cards, optical disk, etc. It is shown by two parallel lines with the name of the data store between them.

Guidelines for Constructing DFDs :

(i) Choose meaningful names for processes, stores, data flows and entities. Each of these should be numbered as well.

(ii) The direction of flow of data is from top to bottom and from left to right. Data flows from the source to the destination, and may flow back to the source as well.

(iii) When a process is broken into lower level processes, each of them must be numbered. For example, if process 5 is divided into sub processes then each of them will be numbered 5.1, 5.2, 5.3, etc.

(iv) The names of data stores, entities are written in capital letters. Process and data flow names must have the first letter of each word in capital.

Context Diagrams : The context diagram represents the entire system under investigation. This diagram should be drawn first, and used to clarify and agree the scope of the investigation. The components of a context diagram are clearly shown on this screen. The system under investigation is represented as a single process, connected to external entities by data flows and resource flows. Data stores are not part of a context model since these would be internal to the system.

After the context model is created the process is exploded to the next level to show the major processes in the system. Depending upon the complexity of the system each of these processes can also be exploded into their own process model. This continues until the goal of each process accomplishing a single function is reached. Because of this approach the context model is referred to as Level 0 (Zero) DFD, the next as Level 1 DFD, etc.

Chapter-5

Software Quality & Testing

Q.1 What is Software Quality Assurance (SQA)?

Ans.: Software Quality Assurance (SQA) consists of a means of monitoring the software engineering processes and methods used to ensure quality. This is done by means of audits of the quality management system under which the software system is created. It is distinct from software quality control which includes reviewing requirement documents, and software testing. SQA encompasses the entire software development process, which includes processes such as software design, coding, source code control, code reviews, change management, configuration management, and release management. Whereas software quality control is a control of products, software quality assurance is a control of processes.

Software quality assurance is related to the practice of quality assurance in product manufacturing. There are, however, some notable differences between software and a manufactured product. These differences stem from the fact that the manufactured product is physical and can be seen whereas the software product is not visible. Therefore its function, benefits and costs are not easily measured. What's more, when a manufactured product rolls off the assembly line, it is essentially a complete, finished product, whereas software is never finished. Software lives, grows, evolves, and metamorphoses, unlike its tangible counterparts. Therefore, the processes and methods to manage, monitor, and measure its ongoing quality are as fluid and sometimes elusive as are the defects that they are meant to keep in check.

Q.2 Write different advantages of SQA.

Ans.: The different advantages of SQA are :

(1) Improved Customer Satisfaction : Improved customer satisfaction means longer, more profitable customer relationships, positive customer testimonials, and waves of referral business generated from positive word of mouth.

If customers are dissatisfied with a product they have purchased from a particular software vendor, they're likely never to recommend that product nor buy from that software vendor again. Bugs and defects, in addition to seriously hampering an application's functionality, look sloppy and unprofessional, and reflect poorly on a company's reputation.

Without proper testing, it is virtually impossible to know how new users will respond to an application's functions, options, and usability features. Unbiased software quality assurance specialists come to a project fresh, with a clear outlook, and so serve as the first line of defense against unintuitive user interfaces and broken application functionality. A quality application is guaranteed to result in enhanced customer satisfaction.

(2) **Reduced Cost of Development :** Because the process of software quality assurance is designed to prevent software defects and inefficiencies, projects that incorporate rigorous, objective testing will find that development costs are significantly reduced since all later stages of the development life cycle become streamlined and simplified. With SQA, all further testing and development including user testing and customer deployments will go more smoothly, and of course more quickly -- which means your software development project will consistently reach completion on time and within budget, release after release.

(3) **Reduced Cost of Maintenance :** Bug-infested applications are troublesome to support. The combined cost of unnecessary recalls, returns, and patches can be frightful. And that says nothing of what will have to be spent on ongoing customer support, be it by telephone, email, or in person. All these costs and more can be dramatically reduced by releasing only rigorously quality-assured products. Software vendors that invest in quality now can avoid big losses in the future.

Q.3 List the different methods used for SQA.

Ans. Different software applications require different approaches when it comes to testing, but some of the most common tasks in SQA include :

(1) **Validation Testing :** Validation testing is the act of entering data that the tester knows to be erroneous into an application. For instance, typing "Hello" into an edit box that is expecting to receive a numeric entry.

(2) **Data Comparison :** Comparing the output of an application with specific parameters to a previously created set of data with the same parameters that is known to be accurate.

(3) **Stress Testing :** A stress test is when the software is used as heavily as possible for a period of time to see whether it copes with high levels of load. Often used for server software that will have multiple users connected to it simultaneously. Also known as Destruction testing.

(4) **Usability Testing :** Sometimes getting users who are unfamiliar with the software to try it for a while and offer feedback to the developers about what they found difficult to do is the best way of making improvements to a user interface.

Q.4 What is a White Box Testing Strategy?

Ans.: White box testing strategy deals with the internal logic and structure of the code. White box testing is also called as glass, structural, open box or clear box testing. The tests written based on the white box testing strategy incorporate coverage of the code written, branches, paths, statements and internal logic of the code etc.

In order to implement white box testing, the tester has to deal with the code and hence is needed to possess knowledge of coding and logic i.e. internal working of the code. White box test also needs the tester to look into the code and find out which unit/statement/chunk of the code is malfunctioning.

Q.5 What are the Advantages & Disadvantages of White box testing?

Ans.: White box testing has the following Advantages & Disadvantages -

Advantages :

ii) As the knowledge of internal coding structure is a prerequisite, it becomes very easy to find out which type of input/data can help in testing the application effectively.

iii) The other advantage of white box testing is that it helps in optimizing the code.

iv) It helps in removing the extra lines of code, which can bring in hidden defects.

Disadvantages :

i) As knowledge of code and internal structure is a prerequisite, a skilled tester is needed to carry out this type of testing, which increases the cost.

ii) It is nearly impossible to look into every bit of code to find out hidden errors, which may create problems, resulting in failure of the application.

Q.6 Define in short the different types of testing under White/Glass Box Testing Strategy.

Ans.: Types of testing under White/Glass Box Testing Strategy -

(1) **Unit Testing :** The developer carries out unit testing in order to check if the particular module or unit of code is working fine. The Unit Testing comes at the very basic level as it is carried out as and when the unit of the code is developed or a particular functionality is built.

(2) **Static and Dynamic Analysis :** Static analysis involves going through the code in order to find out any possible defect in the code. Dynamic analysis involves executing the code and analyzing the output.

(3) **Statement Coverage :** In this type of testing the code is executed in such a manner that every statement of the application is executed at least once. It helps in assuring that all the statements are executed without any side effect.

(4) **Branch Coverage :** No software application can be written in a continuous mode of coding, at some point we need to branch out the code in order to perform a particular functionality. Branch coverage testing helps in validating of all the branches in the code and making sure that no branching leads to abnormal behaviour of the application.

(5) **Security Testing :** Security Testing is carried out in order to find out how well the system can protect itself from unauthorized access, hacking – cracking, any code damage etc. which deals with the code of application. This type of testing needs sophisticated testing techniques.

(6) **Mutation Testing :** A kind of testing in which, the application is tested for the code that was modified after fixing a particular bug/defect. It also helps in finding out which code and which strategy of coding can help in developing the functionality effectively.

Besides all the testing types given above, there are some more types which fall under both Black box and White box testing strategies such as: Functional testing (which deals with the code in order to check its functional performance), Incremental integration testing (which deals with the testing of newly added code in the application), Performance and Load testing (which helps in finding out how the particular code manages resources and give performance etc.).

Q.7 Explain what is Conditional Testing?

Ans.: The first improvement to white box technique is to ensure that the Boolean controlling expressions are adequately tested, a process known as condition at testing.

The process of condition at testing ensures that a controlling expression has been adequately exercised when the software is under test by constructing a constraint set for every expression and then ensuring that every member on the constraint set is included in the values which are presented to the expression. This may require additional test runs to be included in the test plan.

To introduce the concept of constraint sets the simplest possible Boolean condition, a single Boolean variable or a negated Boolean variable, will be considered. These conditions may take forms such as:

if DateValid then while not DateValid then

The constraint set for both of these expressions is {t ,f } which indicates that to adequately test these expressions they should be tested twice with DateValid having the values True and False.

Perhaps the next simplest Boolean condition consists of a simple relational expression of the form value operator value, where the operator can be one of :

- is equal to (=)
- is not equal to (/=)
- is greater than (>)
- is less than (<)
- is greater than or equal to (>=)
- is less than or equal to (<=).

It can be noted that the negation of the simple Boolean variable above has no effect upon the constraint set and that the six relational operators can be divided into three pairs of operators and their negations. Is equal to is a negation of is not equal to, is greater than is a negation of is less than or equal to and is less than is a negation of is greater than or equal to. Thus the condition set for a relational expression can be expressed as {=, >, <}, which indicates that to adequately test a relational expression it must be tested three times with values which ensure that the two values are equal, that the first value is less than the second value and that the first value is greater than the second value.

Thus if only one the left hand Boolean Value is a relational expression the condition set would be {{= ,t } {= ,f } {> ,t } {< ,t } {> ,f } {< ,f }}. And if both Boolean Values are relation expressions this would become {{= ,= } {= ,> } {= ,< } {> ,= } {< ,= } {> ,> } {> ,< } {< ,> } {< ,< }}.

An increase in the complexity of the Boolean expression by the addition of more operators will introduce implicit or explicit bracketing of the order of evaluation which will be reflected in the condition set and will increase the number of terms in the set. For example a Boolean expression of the following form :

BooleanValue1 operator1 BooleanValue2 operator3nbsp;BooleanValue3

Has the implicit bracketing :

(BooleanValue1 operator1 BooleanValue2) operator3 BooleanValue3

The constraint set for the complete expression would be {{e1,t}{e1,f}}, where e1 is the condition set of the bracketed sub-expression and when it is used to expand this constraint set gives {{t,t,t} {t,f,t} {f,t,t} {f,f,t} {t,t,f} {t,f,f} {f,t,f} {f,f,f}}. If any of the BooleanValues are themselves relational expressions this will increase the number of terms in the condition set. In this example the worst case would be if all three values were relational expressions and would produce a total of 27 terms in the condition set. This would imply that 27 tests are required to adequately test the expression. As the number of Boolean operators increases the number of terms in a condition set increases exponentially and comprehensive testing of the expression becomes more complicated and less likely. Keep Boolean control expressions as simple as possible, and one way to do this is to use Boolean variables rather than expressions within such control conditions.

Q.8 Define Data Life Cycle Testing.

Ans.: Data life cycle testing, is based upon the consideration that a variable is at some stage created, and subsequently may have its value changed or used in a controlling expression several times before being destroyed.

This approach to testing requires all possible feasible lifecycles of the variable to be covered while the module is under test. In the case of a Boolean variable this should include the possibility of a Boolean variable being given the values True and False at each place where it is given a value.:

```
~~~ SomeSubProgram( ~~~ ) is
ControlVar : BOOLEAN := FALSE;
begin -- SomeSubProgram
~~~
while not ControlVar loop
~~~
ControlVar := SomeExpression;
end loop;
~~~
end SomeSubProg;
```

In this sketch ~~~ indicates the parts of the subprogram which are not relevant to the lifecycle. In this example there are two places where the variable ControlVar is given a value, the location where it is created and the assignment within the loop. Additionally there is one place where it is used as a control expression. There are two possible lifecycles to consider, one which can be characterised as { f t } indicating that the variable is created with the value False and given the value True upon the first iteration of the loop. The other lifecycle can be characterised as { f, f, ... t }, which differs from the first by indicating that the variable is given the value False on the first iteration, following which there is the possibility of more iterations where it is also given the value False, being given the value True on the last iteration.

Q.9 Define Loop Testing.

Ans.: The final white box consideration which will be introduced is the testing of loops, which have been shown to be the most common cause of faults in subprograms. If a loop, definite or indefinite, is intended to iterate n times then the test plan should include the following seven considerations and possible faults.

All feasible possibilities should be exercised while the software is under test. The last possibility, an infinite loop, is a very noticeable and common fault. All loops should be constructed in such a way that it is guaranteed that they will at some stage come to an end. However this does not necessarily guarantee that they come to an end after the correct number of iterations, a loop which iterates one time too many or one time too few is probably the most common loop fault. Of these possibilities an additional iteration may hopefully cause a CONSTRAINT_ERROR exception to be raised announcing its presence. Otherwise the n - 1 and n + 1 loop faults can be very difficult to detect and correct.

A loop executing zero times may be part of the design, in which case it should be explicitly tested that it does so when required and does not do so when it is not required. Otherwise, if the loop should never execute zero times, and it does, this can also be a very subtle fault to locate. The additional considerations, once, twice and many are included to increase confidence that the loop is operating correctly.

The next consideration is the testing of nested loops. One approach to this is to combine the test considerations of the innermost loop with those of the outermost loop. As there are 7 considerations for a simple loop, this will give 49 considerations for two levels of nesting and 343 considerations for a triple nested loop, this is clearly not a feasible proposition.

What is possible is to start by testing the innermost loop, with all other loops set to iterate the minimum number of times. Once the innermost loop has been tested it should be configured so that it will iterate the minimum number of times and the next outermost loop tested. Testing of nested loops can continue in this manner, effectively testing each nested loop in sequence rather than in combination, which will result in the number of required tests increasing arithmetically (7, 14, 21 ..) rather than geometrically (7, 49, 343 ..).

Q.10 What is Black Box Testing?

Ans.: Black Box Testing is testing without knowledge of the internal working of the item being tested. For example, when black box testing is applied to software engineering, the tester would only know the "legal" inputs and what the expected outputs should be, but not how the program actually arrives at those outputs. It is because of this that black box testing can be considered testing with respect to the specifications, no other knowledge of the program is necessary. For this reason, the tester and the programmer can be independent of one another, avoiding programmer bias towards his own work. Also, due to the nature of black box testing, the test planning can begin as soon as the specifications are written.

Q.11 Write the advantages and disadvantages of Black Box Testing.

Ans.: Following are the different **advantages of black box testing -**

- more effective on larger units of code than glass box testing.
- tester needs no knowledge of implementation, including specific programming languages.
- tester and programmer are independent of each other.
- tests are done from a user's point of view.
- will help to expose any ambiguities or inconsistencies in the specifications.
- test cases can be designed as soon as the specifications are complete.

Disadvantages of Black Box Testing -

- only a small number of possible inputs can actually be tested, to test every possible input stream would take nearly forever.
- without clear and concise specifications, test cases are hard to design.
- there may be unnecessary repetition of test inputs if the tester is not informed of test cases the programmer has already tried.
- may leave many program paths untested.
- cannot be directed toward specific segments of code which may be very complex (and therefore more error prone).
- most testing related research has been directed toward glass box testing.

Q.12 Explain the different types of data used in Black Box Technique.

Ans.: In this technique, we do not use the code to determine a test suite; rather, knowing the problem that we're trying to solve, we come up with four types of test data:

1. Easy-to-compute data
2. Typical data
3. Boundary / extreme data
4. Bogus data

For example, suppose we are testing a function that uses the quadratic formula to determine the two roots of a second-degree polynomial ax2+bx+c. For simplicity, assume that we are going to work only with real numbers, and print an error message if it turns out that the two roots are complex numbers (numbers involving the square root of a negative number).

We can come up with test data for each of the four cases, based on values of the polynomial's discriminant (b2-4ac):

Easy Data (discriminant is a perfect square) :

a	b	c	Roots
1	2	1	-1, -1
1	3	2	-1, -2

Typical Data (discriminant is positive) :

a	b	c	Roots
1	4	1	-3.73205, -0.267949
2	4	1	-1.70711, -0.292893

Boundary / Extreme Data (discriminant is zero) :

a	b	c	Roots
2	-4	2	1, 1
2	-8	8	2, 2

Bogus Data (discriminant is negative, or a is zero) :

a	b	c	Roots
1	1	1	square root of negative number
0	1	1	division by zero

As with glass-box testing, you should test your code with each set of test data. If the answers match, then your code passes the black-box test.

Q.13 Define Program Complexity Analysis.

Ans.: Cyclomatic complexity is a software metric (measurement). It was developed by Thomas McCabe and is used to measure the complexity of a program. It directly measures the number of linearly independent paths through a program's source code.

The concept, although not the method, is somewhat similar to that of general text complexity measured by the Flesch-Kincaid Readability Test.

Cyclomatic complexity is computed using a graph that describes the control flow of the program. The nodes of the graph correspond to the commands of a program. A directed edge connects two nodes if the second command might be executed immediately after the first command.

$$M = E - N + 2P$$

Where -

M = cyclomatic complexity

E = the number of edges of the graph

N = the number of nodes of the graph

P = the number of connected components.

"M" is alternatively defined to be one larger than the number of decision points (if/case-statements, while-statements, etc) in a module (function, procedure, chart node, etc.), or more generally a system.

Separate subroutines are treated as being independent, disconnected components of the program's control flow graph.

Q.14 What is Debugging?

Ans.: Debugging occurs as an end result of successful testing. We can say it is a process which removes the errors. This process attempt to match symptom with cause, which leads to error correction. It is an iterative process.

Debugging process always has two outcomes –

(i) Case will be found & corrected.

(ii) The cause will not be found. In this case the debugger may suspect a cause & can design a test case.

Q.15 List the limitations of Debugging.

Ans.: (i) The symptom & the cause could geographically remote.

(ii) The symptom may temporarily disappear when any other error is corrected.

(iii) The symptoms may cause due to inaccuracies.

(iv) The errors could be result of timing problem rather than processing one.

(v) It is difficult to accurately reproduce actual input conditions.

(vi) The symptom could be intermittent between hardware & software in embedded systems.

(vii) Errors could be due to the different tasks running.

Q.16 Define the different approaches used in debugging.

Ans.: In general there are three categories –

(i) **Brute Force :** It is the most common & least efficient method for removing errors. We use it when all else fails.

It works on "Let the computer find the error" philosophy. Here run time traces are invoked & it is hoped that mass of information could give clues to find errors.

(ii) **Back Tracking :** It can be used for small programs. Here the source code is tracked backward manually, until the site of cause is found.

(iii) **Cause Elimination :** Here a binary partition is created. Data related to error occurrence are organized. A "cause hypothesis" is devised. A list of all possible reasons is created & tests are conducted to eliminate them.

□ □ □

Chapter-6

Software Cost & Time Estimation

Q.1 Define the Analogy Costing Method.

Ans.: **Analog Costing :** This method requires one or more completed projects that are similar to the new project and derives the estimation through reasoning by analogy using the actual costs of previous projects. Estimation by analog can be done either at the total project level or at subsystem level. The total project level has the advantage that all cost components of the system will be considered while the subsystem level has the advantage of providing a more detailed assessment of the similarities and differences between the new project and the completed projects. The strength of this method is that the estimate is based on actual project experience. However, it is not clear to what extent the previous project is actually representative of the constraints, environment and functions to be performed by the new system. Positive results and a definition of project similarity in terms of features were reported.

Q.2 Define Delphi Techniques.

Ans.: **Delphi Technique** works as follows :-

(i) The coordinator presents each expert with a specification and a form to record estimates.

(ii) Each experts fills in the form individually (without discussing with others) and is allowed to ask the coordinator questions.

(iii) The coordinator prepares a summary of all estimates from the experts (including mean or median) on a form requesting another iteration of the expert's estimates and the rationale for the estimates.

(iv) Repeat steps second & third as many rounds as appropriate.

A modification of the Delphi technique proposed by Boehm and Fahquhar seems to be more effective : Before the estimation, a group meeting involving the coordinator and experts is arranged to discuss the estimation issues. In step (iii), the experts do not need to give any rationale for the estimates.

Instead, after each round of estimation, the coordinator calls a meeting to have experts discussing those points where their estimates varied widely.

Q.3 Define Putnam / SLIM Model.

Ans.: The **Putnam Model** is an empirical software effort estimation model. As a group, empirical models work by collecting software project data (for example, effort and size) and fitting a curve to the data. Future effort estimates are made by providing size and calculating the associated effort using the equation which fit the original data (usually with some effort). Created by Lawrence Putnam, Sr. the **Putnam Model** describes the time and effort required to finish a software project of specified size. SLIM (Software Lifecycle Management) is the name given by Putnam to the proprietary suite of tools his company QSM. Inc. has developed based on his model. It is one of the earliest of these types of models developed, but is not the most widely used.

Putnam's Model and SLIM :

Putnam derives his model based on Norden/Rayleigh manpower distribution and his finding in analyzing many completed projects. The central part of Putnam's model is called software equation as follows :

$$S = E \times (Effort)^{1/3} td^{4/3}$$

Where 'td' is the software delivery time; E is the environment factor that reflects the development capability, which can be derived from historical data using the software equation. The size S is in LOC and the Effort is in person-year. Another important relation found by Putnam is

$$Effort = D_0 \times td^3$$

Where D_0 is a parameter called manpower built-up which ranges from 8 (entirely new software with many interfaces) to 27 (rebuilt software).

Combining the above equation with the software equation, we obtain the power function form :

$$Effort = (D_0^{4/7} \times E^{-9/7}) \times S^{9/7} \qquad \text{and}$$

$$td = (D_0^{-1/7} \times E^{-3/7}) \times S^{3/7}$$

Putnam's Model is also widely used in practice and SLIM is a software tool based on this model for cost estimation and manpower scheduling.

Q.4 Explain Reyleigh Distribution Curve.

Ans.: The Rayleigh Distribution Curve slopes upward, levels off into a plateau, and then tails off gradually. It is described by the equation :

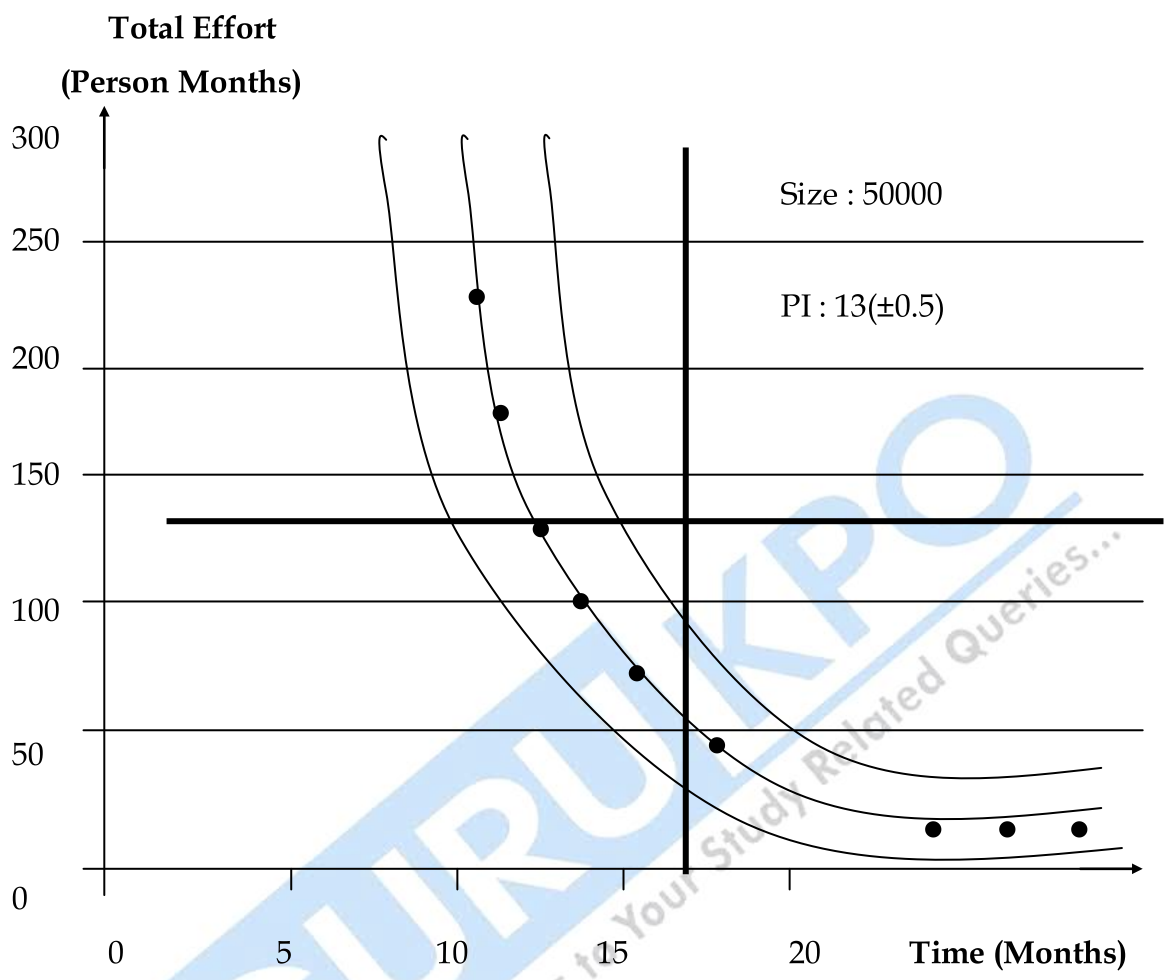
$$y = 2Kate^{-at2}$$

where

- y = manpower in man-years per year (MY/YR)
- K = total software effort (equivalent to the total area under the curve)
- $a = 1/(2t_d^{\,2})$
- t_d = time of maximum manpower
- t = instantaneous time

According to Putnam, the Raleigh curve depicts the profile of a software development project, with time on the horizontal axis and manpower on the vertical axis.

— Effort (Person Months) — ±1 Dev. — Effort Limit — Time Limit



Based on this equation, Putnam states "... if we know the management parameters K and td, then we can generate the manpower, instantaneous cost, and cumulative cost of a software project at any time t by using the Rayleigh equitation."

Putnam used his observations about productivity levels to derive the software equation :

$$S = E * (Effort)^{1/3} t_d^{4/3}$$

Where :

- E is the environment factor that reflects the development capacity.
- Efforts is the total effort applied to the project in person-years.
- t_d is the software delivery time.
- S is the size in LOC.

Q.5 Describe Walston and Felix Model.

Ans.: Walston and Felix performed some of the early work that led to the first generation of software effort estimation techniques. In one particular paper, they collected and analyzed numerous measurable parameters from sixty completed projects in order to arrive at empirically-determined estimation equation. For example,

$$E = 5.2L^{0.91}$$

(where E = total effort in man-months (MM) and L = thousands of lines of delivered source code).

This power relationship between effort and program size resembles the result derived by other investigators such as Nelson, Freburger-Basili, and Herb, who formulated the following relations, respectively : $E = 4.9L^{0.98}$, $E = 1.48L^{1.02}$, and $E = 5.3L^{1.06}$.

Walston and Felix also derived an equation for average staff size :

$$S = 0.54E^{0.6}$$

(where S = average project staff size, and E = total effort).

Theoretically, one could estimate project parameters such as effort, average staff size, as well as total costs by simply estimating the total lines of code and using the appropriate equations.

□ □ □

BACHELOR OF COMPUTER APPLICATIONS (Part III) EXAMINATION

(Faculty of Science)

(Three – Year Scheme of 10+2+3 Pattern)

PAPER 312

Software Engineering

OBJECTIVE PART- I

Year - 2011

Time allowed : One Hour *Maximum Marks : 20*

The question paper contains 40 multiple choice questions with four choices and students will have to pick the correct one. (Each carrying ½ marks.).

1. SDLC stands for:
   (a) System Development Life Cycle
   (b) Software Development Life Cycle
   (c) Systematic Development Life Cycle
   (d) Solution Development Life Cycle ( )

2. Software development environment that supports the 4 GT paradigm includes:
   (a) Non-procedural language for database query
   (b) Spreadsheet capability
   (c) Screen interaction and definition
   (d) All of the above ( )

3. Project Risk factor is incorporated in:
   (a) Spiral Model
   (b) Iterative enhancement model
   (c) Evolutionary enhancement model
   (d) Waterfall model ( )

4. Which one is not a phase is "Waterfall model"?
   (a) Design
   (b) Code
   (c) Maintenance
   (d) Recovery ( )

5. In DFD, rectangle is used to represent:
   (a) Processes (b) Flow of data
   (c) Source data (d) None of the above ( )

6. Entity relationship is used in:
   (a) Data design
   (b) Architectural design
   (c) Interface design
   (d) None of the above ( )

7. PERT is:
   (a) Prototyping tool
   (b) Re-engineering tool
   (c) Change Management Tool
   (d) Planning tool ( )

8. Which is not an element of software matrices domain?
   (a) Productivity
   (b) Quality
   (c) Availability
   (d) Human ( )

9. How many levels are there is CMM?
   (a) One
   (b) Three
   (c) Four
   (d) Five ( )

10. Which is not the decomposition technique?
    (a) Software sizing (b) Feasibility
    (c) Process based estimation (d) FP based estimation ( )

11. Fundamental approach to identify the test cases are:
    (a) Functional Testing
    (b) Structural Testing
    (c) Both (a) and (b)
    (d) None of the above ( )

12. Classical integration strategies are:
    (a) Top-down integration (b) Bottom-up integration
    (c) Sandwich integration (d) All of the above ( )

13. System testing does not include:
    (a) Recovering testing

(b) Alpha testing
(c) Stress testing
(d) Grey testing ( )

14. Method of equivalence partitioning is:
(a) White box testing
(b) Black box testing
(c) Both (a) and (b)
(d) Security testing ( )

15. The classic life-cycle model is also known is:
(a) Linear sequential model
(b) Prototyping model
(c) Spiral model
(d) None of the above ( )

16. Which model is complete in very short period ?
(a) Spiral model
(b) Waterfall model
(c) RAD model
(d) All of the above ( )

17. Which of the following testing methods is normally used as the acceptance test for a S/W system?
(a) Regression testing
(b) Integration testing
(c) Unit testing
(d) Functional testing ( )

18. An important aspect in coding is:
(a) Readability
(b) Productivity
(c) Brevity
(d) None of the above ( )

19. Software engineering primarily aims on:
(a) Reliable software
(b) Cost effective software
(c) Both (a) and (b)
(d) None of the above ( )

20. Structure programming code includes:
(a) Sequencing
(b) Alteration
(c) Iteration

(d) All of the above ( )

21. Which of the following is a tool design phase?
(a) Abstraction
(b) Refinement
(c) Information hiding
(d) All of the above ( )

22. Primary advantages of the object oriented approach is:
(a) Increased productivity
(b) Ease of testing and maintenance
(c) Better code and design understandability
(d) All of the above ( )

23. Which of the following graph theoretic concept will be useful in software testing?
(a) Cyclometric number
(b) Hamiltonian
(c) Eulerian cycle
(d) None of the above ( )

24. HIPO charts facilitate:
(a) Analysis of modules
(b) Top down design
(c) Organization structure
(d) None of the above ( )

25. Which is the most critical phase of the SDLC?
(a) Feasibility study
(b) System Analysis
(c) System design
(d) All of the above ( )

26. What are the four P's of software engineering according to Braude?
(a) People, Product, Project, Private
(b) People, product, project personnel
(c) People, product, project process
(d) None of the above ( )

27. SRM stands for:
(a) Software Resources Management
(b) Software Risk Management
(c) Software Risk Measurement
(d) None of the above ( )

28. Matrices are...............collection of data, resources and deliverable about project activities.
    (a) Measurement
    (b) Observation
    (c) Processes
    (d) None of the above ( )

29. A module consist of:
    (a) Single entry only
    (b) Single entry and single exist only
    (c) Multiple entries and single exist only
    (d) Multiple entries and multiple exists only ( )

30. The interface design of software includes:
    (a) Data Dictionary
    (b) E-R diagram
    (c) Data flow diagram
    (d) State transition diagram ( )

31. The relative functional strength of a model depends on:
    (a) Coupling (b) Coupling and cohesion
    (c) Cohesion (d) Expendability ( )

32. "Verification and validation" falls under whose roles?
    (a) Designing team
    (b) Specification team
    (c) Documentation team
    (d) Quality assurance team ( )

33. 'CASE' stands for:
    (a) Computer accessibility and software environment
    (b) Computer Aided Software Environment
    (c) Computer Aided software Engineering
    (d) Computer Aided System Engineering ( )

34. Waterfall model is best suited for S/W with:
    (a) Rigid Requirement
    (b) Flexible Requirement
    (c) Changeable Requirements
    (d) Requirement are not well understood ( )

35. Which of the following is non-functional require meant?
    (a) Execution time (b) Searching capability

(c) Can put data to database (d) Customized user interface ( )

36. Which model reinforces the notion of 'Define before Design' and Design before code?
(a) Build and Fix
(b) Prototyping Model
(c) Classical Life Cycle
(d) Spiral Model ( )

37. The importance of software design can be summarized in a single word:
(a) Accuracy
(b) Complexity
(c) Efficiently
(d) Quality ( )

38. Control flow diagram are:
(a) Needed to model event driven systems
(b) Required for all system
(c) Useful for model Ingra time system
(d) Cyrus-Beck Algorithm ( )

39. Which of the following need to be assessed during unit testing?
(a) Algorithmic performance
(b) Code stability
(c) Error-handling
(d) Error handling and execution paths ( )

40. Acceptance tests are normally conducted by the :
(a) Developer
(b) End users
(c) Test team
(d) System Engineers ( )

DESCRIPTIVE PART-II

Year- 2011

Time allowed : 2 Hours ***Maximum Marks : 30***

Attempt any four descriptive types of questions out of the six. All questions carry 7½ marks each.

1. What are the issued in the design of the software? Explain its need for the software engineering?

2. Discuss the prototype model. What is the effect of designing a prototype on the overall cost of the software project?

3. What is software project planning. Write a note on the approaches suggest by Putnam and Myers to the sizing problem.

4. What are the objectives of testing? Write down the testing principles. Write a note on white box testing.

5. Differentiate between "Verification " and "Validation" Explain verification and validation Testing techniques.

6. Write short notes on:
   (i) Software Requirement Specification
   (ii) COCOMO model;
   (iii) E-R Diagram
   (iv) Fourth Generation Techniques
   (v) Data Dictionary;

BACHELOR OF COMPUTER APPLICATIONS

(Part III) EXAMINATION

(Faculty of Science)

(Three – Year Scheme of 10+2+3 Pattern)

PAPER 312

Software Engineering

OBJECTIVE PART- I

Year - 2010

Time allowed : One Hour ***Maximum Marks : 20***

The question paper contains 40 multiple choice questions with four choices and students will have to pick the correct one. (Each carrying ½ marks.).

1. Operating System fall under which category of software?
   (a) Middleware (b) Embedded
   (c) Utility (d) System ( )

2. Software Engineering is practised through:
   (a) SSAD and OOSAD (b) SAD adn SSD
   (c) OOP's and SSAD (d) OOp's and OOSAD ( )

3. The components of a project include:
   (a) Products
   (b) People
   (c) Process and Property
   (d) All of the above ( )

4. The creation and reuse of Software building block is:
   (a) Usability (b) Reusability
   (c) Adaptability (d) Non-adaptability ( )

5. The RAD Model Stands for:
   (a) Rapid Application Development (b) Role Action Development
   (c) Rapid and Direct (d) Reuse Action Development ( )

6. Which Software Development model takes more development time?
   (a) Prototype Model (b) Waterfall Modal
   (c) Incremental Model (d) Spiral Model ( )

7. State which one is not 4GL techniques?
   (a) Report generation (b) Data manipulation
   (c) Code generation (d) Programming technique ( )

8. Which of most critical phase of the SDLC?
   (a) Feasibility Study (b) System Analysis
   (c) System Design (d) All of the above ( )

9. Which model is completed in very short period?
   (a) Spiral Model (b) Waterfall Model
   (c) RAD Model (d) All of the above ( )

10. A good specification should be:
    (a) Unambiguous (b) Distinctly specific
    (c) (a) and (b) both (d) None of the above ( )

11. ....feasibility involves a study of functions, performance and constraints that may affect the ability to achieve an acceptable system;
    (a) Economic
    (b) Technical
    (c) Legal
    (d) Business ( )

12. In the basic COCOMO model effort depends on size, which is in terms of :
    (a) LOC (b) FP
    (c) Time (d) All of the above ( )

13. Which is the most appropriate unit of 'effort' as on COCOMO mode?
    (a) Persons (b) Person/Months
    (c) Person Month (d) Months ( )

14. ....is a measure of length of code the software engineering will write to deliver software requirement.
    (a) LOC (b) LCD
    (c) LHC (d) LHD ( )

15. Which class of Software Project belongs to COCOMO model?
    (a) Organic (b) Physical
    (c) Non-physical (d) Inorganic ( )

16. Which is the full form of COCOMO model?
    (a) Concurrent Cost Model
    (b) Constructive Cost Model
    (c) Concurrency Cost Model
    (d) Collecting Cost Model ( )

17. Which of the following shows the system behavior with the external events?
(a) E-R diagrams (b) Data flow diagrams
(c) State transition (d) Data Dictionary ( )

18. ....is a collection of class definitions.
(a) Packages (b) Interfaces
(c) Libraries (d) Components ( )

19. Primary advantage of the object-oriented approach is :
(a) Increase productivity
(b) Ease of testing and maintenance
(c) Better code and design understandability
(d) All of the above ( )

20. Inheritance defines as:
(a) A new class by extending an existing class
(b) Object constitute a class
(c) An object which interfaces with outside world
(d) Same message can result in different actions ( )

21. Which one is not an essential part of OOAD?
(a) Class diagram (b) Object diagram
(c) Sequence diagram (d) ER diagram ( )

22. Software design concept is related to :
(a) Algorithms (b) Flow Charts
(c) Modularity (d) Reliability ( )

23. The relative functional strength of a module depends on:
(a) Coupling
(b) Cohesion
(c) Coupling & Cohesion
(d) Expendability ( )

24. Data elements definitions includes:
(a) Field Name (a) (b) Detailed description
(c) Databases (d) Data Stores ( )

25. Structured design is a ....methodology:
(a) Data flow based (b) Data driven based
(c) Data decomposition based (d) Data structure based ( )

26. What concepts of object oriented programming hides the information outside world:
(a) Inheritance (b) Polymorphism

(c) Abstraction (d) Dynamic binding ( )

27. Structured programming Code includes:
(a) Sequencing
(b) Alteration
(c) Iteration
(d) All of the above ( )

28. Design phase will usually be:
(a) Top-down
(b) Bottom-up
(c) Random
(d) Interface design ( )

29. Assertions are conditions which are true at the point of execution:
(a) Always
(b) Sometimes
(c) Many times
(d) No time ( )

30. Data structure suitable for the application is discussed in:
(a) Data design (b) Architectural Design
(c) Procedural design (d) Interface desing ( )

31. Which of the following statement is not true?
(a) Content coupling in a module is desirable
(b) Logical Cohesion in a module is desirable
(c) Stamp coupling in a module is desirable
(d) All of the above ( )

32. An import aspect in coding is :
(a) Readability
(b) Productivity
(c) Brevity
(d) To use a small memory space as possible ( )

33. Unit testing mostly deals in:
(a) Modular testing (b) Procedural testing
(c) Architectural testing (d) None of the above ( )

34. Software testing is performed for :
(a) Verification only
(b) Validation only
(c) Verification and Validation both
(d) System designing only ( )

35. System testing does not include:
    (a) Recovering testing
    (b) Alpha testing
    (c) Stress testing
    (d) Security ( )

36. White box testing is also knows as:
    (a) Behavioural testing
    (b) Glass box testing
    (c) Architectural testing
    (d) None of the above ( )

37. 'Cyclometric Complexity' and ' Basis path' testing full under which category of testing:
    (a) Black box
    (b) White box
    (c) Both (a) and (b)
    (d) None of the above ( )

38. Software Re-engineering is also called as:
    (a) Software renewal (b) Software Analysis
    (c) System Design (d) None of the above ( )

39. ....is the last phase before the final software is delivered:
    (a) Analysis
    (b) Testing
    (c) Design
    (d) None of the above ( )

40. Verification and Validation fall under whose roles:
    (a) Designing team
    (b) Specification team
    (c) Documentation team
    (d) Quality assurance team ( )

Answer Key

1. (d)	2. (a)	3. (d)	4. (b)	5. (a)	6. (c)	7. (d)	8. (d)	9. (c)	10. (c)
11. (b)	12. (a)	13. (c)	14. (a)	15. (a)	16. (b)	17. (c)	18. (a)	19. (d)	20. (a)
21. (d)	22. (c)	23. (c)	24. (b)	25. (c)	26. (c)	27. (d)	28. (a)	29. (a)	30. (a)
31. (a)	32. (a)	33. (a)	34. (c)	35. (b)	36. (b)	37. (b)	38. (a)	39. (b)	40. (d)

DESCRIPTIVE PART-II

Year- 2010

Time allowed : 2 Hours ***Maximum Marks : 30***

Attempt any four descriptive types of questions out of the six. All questions carry 7½ marks each.

Q.1 What is "Software Engineering' ? State the difference between software Engineering and Traditional Engineering.

Q.2 Why is Management Process important in a software Project? Illustrate the relationship between 'Development Process' and ' Management Process".

Q.3 What do you mean by Software 'Quality' ? What are various Software quality attributes?

Q.4 Does simply presence of fault mean Software failure? If no, justify your answer with proper example.

Q.5 What is 'design review' ? How design review can uncover deficiencies in SRS? Explain 'Coupling' and 'Cohesion' in brief.

Q.6 Writes short notes on any three:
(a) SRS
(b) Cost Estimation Model
(c) Process metrics Vs. Product metrics

OBJECTIVE PART- I

Year - 2009

Time allowed : One Hour *Maximum Marks : 20*

The question paper contains 40 multiple choice questions with four choices and student will have to pick the correct one. (Each carrying ½ marks.).

1. Which of the items listed below is not one of the software engineering layers?
(a) Process (b) Manufacturing
(c) Methods (d) Tools ( )

2. Which of the following are the 5 generic software engineering framework activities?
(a) Communication, planning, modeling, construction, deployment
(b) Communication, risk management, measurement, production, reviewing
(c) Analysis, designing, programming, debugging, maintenance
(d) Analysis, planning, Designing, programming, testing ( )

3. Software processes can be constructed out pre-existing software patterns to best beet the need of a software project.
(a) True (b) False
( )

4. Which of these are standards for assessing software processes?
(a) SEI (b) Spice
(c) ISO9001 (d) Both B and C ( )

5. The linear sequential model of software development is also known as the:
(a) Classical life cycle model (b) Fountain model
(c) Waterfall model (d) Both A and B ( )

6. The rapid application model is:
(a) Another name for component-based development
(b) A useful approach when a customer can not define requirements clearly
(c) A high speed adaption of the linear sequential model
(d) All of the above ( )

7. Evolutionary software process models:
(a) Are iterative in nature
(b) Can easily accommodate product requirement change
(c) Do not generally produce throw away systems

(d) All of the above ( )

8. The prototyping model of software development is:
   (a) A reasonable approach when requirement are well defined
   (b) A useful approach when a customer can not large development teams
   (c) The best approach to use for project with large development teams
   (d) A risky mode that producers a meaningful product ( )

9. The spiral model of software development is:
   (a) Ends with the delivery of the software product
   (b) Is more chastic than the incremental model
   (c) Includes project risks evaluation during each iteration
   (d) All of the above ( )

10. The component based development model is:
   (a) Only appropriate for computer hardware design
   (b) Net able to support the development of reusable components
   (c) Works best when object technologies are available for support
   (d) Not cost effective by known quantifiable software metrics ( )

11. Which of these is not one of the phase names defined by the unified process model for software development.
   (a) Inception phase
   (b) Elaboration phase
   (c) Construction phase
   (d) Validation phase ( )

12. Which of the following is not necessary to apply agility to a software process:
   (a) Eliminate the use of project planning and testing
   (b) Only essential work products produced
   (c) Process allows team to streamline tasks
   (d) Uses incremental product delivery strategy ( )

13. In agile software processes the highest priority is to satisfy the customer through early and continuous delivery of valuable software.
   (a) True
   (b) False ( )

14. What are the three framework activities for the adaptive Software Development (ASD) process model?
   (a) Analysis, design coding
   (b) Feasibility study, functional design, implementation
   (c) Requirement gathering, adaptive cycle planning iterative development
   (d) speculation, collaboration learning ( )

15. Agile modeling (AM) provides guidance to practitioner during which of these software tasks ?
    (a) Analysis (b) Design
    (c) Coding (d) Testing ( )
    (e) Both a and b

16. Software engineers collaborate with customers to define which of the following:
    (a) Customer visible usages scenarios
    (b) Important software outputs
    (c) System inputs and output
    (d) All of the above ( )

17. Everyone in the software team should be involved in the planning activity so that we can:
    (a) Reduce the granularity of the plan
    (b) Analyze requirement in depth
    (c) Get all team member to 'sing up" to the plan
    (d) Begin design ( )

18. Analysis models depict software in which three representations:
    (a) Architecture, interface, component
    (b) cost, risk, schedule
    (c) Information, function, behavior
    (d) None of the above ( )

19. Which of the following is not one of the principles of good coding?
    (a) Create unit tests before you begin coding
    (b) Create a visual layout that aids understanding
    (c) Keep variable names short so that code is compact
    (d) Write soft documenting code, not program documentation ( )

20. Which of the following are valid reasons for collecting customer feedback concerning delivered software?
    (a) Allows developers to made change to the delivered increment
    (b) Delivery schedule can be revised to reflect changes
    (c) Developers can identify to incorporate into next increment
    (d) All of the above ( )

21. The system engineering process usually begins with the:
    (a) Detailed view (b) Domain view
    (c) Element view (d) World view ( )

22. The top level of the hierarchical model of a system is known as the:
    (a) AFD (b) DFD
    (c) SCD (d) SFS ( )

23. The system model template contains which of the following elements?
    (a) Input
    (b) Output
    (c) User interface
    (d) All of the above ( )

24. UML notations that can be used to model the hardware and software elements of a system are:
    (a) Activity diagrams (b) Class diagrams
    (c) Deployment diagrams (d) All of the above ( )

25. The results of the requirement engineering elaboration task is an analysis model that defines which of the following problem domain(s)?
    (a) Information
    (b) Functional
    (c) Behavrioual
    (d) All of the above ( )

26. Which of following is not a UML diagram used creating a system analysis model?
    (a) Activity diagram (b) Class diagram
    (c) Dataflow diagram (d) State diagram ( )

27. The data dictionary contains descriptions of each software:
    (a) Control item
    (b) Data Object
    (c) Notation
    (d) Both a and b ( )

28. Which of these is not an elements of an object oriented analysis model?
    (a) Behavrioural elements
    (b) Class based elements
    (c) Data elements
    (d) Scenario-based elements ( )

29. A generalized description of a collection of similar object is a:
    (a) Class
    (b) Instance
    (c) Subclass
    (d) Super class ( )

30. Control flow diagrams are:
    (a) Needed to model event driven system
    (b) Required for all system
    (c) Useful for modeling real time systems
    (d) Both a and c ( )

31. The importance of sotfware design can be summarized in a single word:
    (a) Accuracy
    (b) Complexity
    (c) Efficiency
    (d) Quality ( )

32. Cohesion is a qualitative indication of the degree to which a module?
    (a) Can be written more compactly
    (b) Focus on just one thing
    (c) Is able to complete its function in a timely manner
    (d) Is connected to there modules and the outside world ( )

33. Coupling is a qualitative indication of the degree to which a module:
    (a) Can be written more compactly
    (b) Focuses on just one things
    (c) Is able to complete its function in a timely manner
    (d) Is connected to there modules and the outside world ( )

34. The best reason for using independent software test teams is that:
    (a) Software developers do not need to do any testing
    (b) A test team will test the software more thoroughly
    (c) Testers do not get involved with the project until testing begins
    (d) Arguments between developers and testers reduced ( )

35. What is the normal order of activities in which traditional 'software testing'.
    (a) Integration testing (b) System testing
    (c) Unit testing (d) Validation testing

    (a) A,C,C,B (b) B,D,A,C
    (c) C,A,D,B (d) D,B,C,A ( )

36. Which of the following need to be assessed during unit testing?
    (a) Algorithmic performance
    (b) Error handling
    (c) Error handling
    (d) Error handling and execution paths ( )

37. Bottom up integration testing has as 't's major advantages that:
    - (a) Major decision points are tested early
    - (b) No drivers need to be written
    - (c) No stubs need to be written
    - (d) Regression testing is not required ( )

38. Regression testing should be a normal part of integration testing because as a new module is added to the system new:
    - (a) Control logic is invoked
    - (b) Data flow paths are established
    - (c) Drivers require testing
    - (d) Both a and B ( )

39. Smoke testing might best be described as:
    - (a) Bullet proofing shrink wrapped software
    - (b) Rolling integration testing
    - (c) Testing that hides implementation error
    - (d) Unit testing for small programs ( )

40. Acceptance tests are normally conducted by the :
    - (a) Developer
    - (b) End users
    - (c) Test team
    - (d) Ststems engineers ( )

Answer Key

1. (b)	2. (a)	3. (b)	4. (d)	5. (d)	6. (c)	7. (d)	8. (b)	9. (d)	10. (d)
11. (d)	12. (a)	13. (a)	14. (d)	15. (a)	16. (d)	17. (a)	18. (c)	19. (a)	20. (d)
21. (d)	22. (c)	23. (d)	24. (d)	25. (d)	26. (d)	27. (d)	28. (c)	29. (a)	30. (a)
31. (d)	32. (b)	33. (d)	34. (b)	35. (c)	36. (d)	37. (c)	38. (d)	39. (a)	40. (c)

DESCRIPTIVE PART - II

Year 2009

Time allowed : 2 Hours ***Maximum Marks : 30***

Attempt any four questions out of the six. All questions carry 7½ marks each.

Q.1 What are the issues in the design of the software? Explain it's need for the software engineering?

Q.2 Explain spiral model in detail with advantage and disadvantages in detail.

Q.3 What are various activities during software project planning? describe any two software size estimation techniques.

Q.4 Explain the prototype model of SDLC?

Q.5 What is the difference between white box and black box testing? Will exhaustive testing guarantee that the program is 100% correct?

Q.6 Write short note on any three:
(a) Data dictionaries
(b) Configuration management
(c) Change control process
(d) SRS

OBJECTIVE PART- I

Year - 2008

Time allowed : One Hour ***Maximum Marks : 20***

The question paper contains 40 multiple choice questions with four choices and student will have to pick the correct one. (Each carrying ½ marks.).

1. Software means:
   (a) Programs
   (b) Operating Procedure
   (c) Documentation
   (d) Collection of all the above ( )

2. Which is not a part of user manual?
   (a) Installation Guide (b) Reference Guide
   (c) System Overview (d) Beginner's Guide Tutorial ( )

3. SDLC stands for:
   (a) System Development Life Cycle
   (b) Software development Life Cycle
   (c) Systematic Development Life Cycle
   (d) Solution Development Life Cycle ( )

5. Which model reinforces the notion of 'Define before Desing" and "Design before code'?
   (a) Build & Fix
   (b) Prototyping Model
   (c) Classical Life Cycle
   (d) Spiral Model ( )

5. Project Risk factor is incorporated in :
   (a) Spiral Model
   (b) Iterative Enhancement Model
   (c) Evolutionary Enhancement Model
   (d) Waterfall model ( )

6. Capability Maturity Model is a:
   (a) Software Life Model

(b) Linear Sequential Model
(c) Attempt to improve the software quality
(d) Strategy for improving the software process ( )

7. Software development environment the support the 4GT paradigm includes:
(a) Non procedural capability
(b) Spreadsheet capability
(c) Screen interaction and definition
(d) All of the above ( )

8. Effective software project management focuses on:
(a) People and product (b) Process and project
(c) Both a and b (d) None of the above ( )

9. Software project planning involves estimation attempt to determine:
(a) How much money and effort
(b) How much resources and time
(c) Both a and b
(d) None of the above ( )

10. COCOMO stands for:
(a) Constructive cost model
(b) Cumulative cost model
(c) Comparative cost model
(d) None of the above ( )

11. Which approach is not suggested by Putnam and myers for sizing problems:
(a) Fuzzy logic sizing
(b) Function point sizing
(c) Sizing of project deliverables
(d) Standard component sizing ( )

12. SDD stand for:
(a) Software dependent document
(b) Software design document
(c) Software Diagram Document
(d) None of the above ( )

13. Objective of design is to specify the:
(a) What
(b) How
(c) When
(d) All of the year ( )

14. Various terms related to object oriented design are:

(a) Object & Classes
(b) Abstraction & Inheritance
(c) Both a and b
(d) None of the above ( )

15. Fundamental approach to identify the test cases are:
(a) Functional testing (b) Structural Testing
(c) Both a and b (d) None of the above ( )

16. Behavrioual testing is also known as:
(a) White Box Testing (b) Glass Box Testing
(c) Structural Testing (d) Black Box Testing ( )

17. Verification means:
(a) Are we building the product right
(b) Are we building the right product
(c) Both a and b
(d) None of the above ( )

18. Validation means:
(a) Are we building the right product
(b) Are we building the product right
(c) Both a and b
(d) None of the above ( )

19. Classical integration strategies are:
(a) Top - down integration (b) Bottom- up integration
(c) Sandwich integration (d) All of the above ( )

20. The design needs to be:
(a) Correct and incomplete (b) Correct and complete
(c) Incorrect and complete (d) Incorrect and incomplete ( )

21. 4 GT begins with the :
(a) Designing (b) Implementation
(c) Requirement gathering (d) None of the above ( )

22. How many levels are in CMM:
(a) One (b) Three
(c) Four (d) Five ( )

23. Readability in coding can be increase by:
(a) Avoiding goto statement
(b) Avoiding nested if statement

(c) Name variables and function according to their use
(d) All of the above ( )

24. Estimation of resources is must in project planning. The statement is :
(a) True (b) False
(c) Not Necessary (d) None ( )

25. Estimation of resources in project covers the estimation for:
(a) Human Resource
(b) Reusable software Resources
(c) Hardware/Software
(d) All of the above ( )

26. Which is not the decomposition technique?
(a) Software sizing (b) Feasibility
(c) Process based estimation (d) FP based estimation ( )

27. COCOMO- II is hierarchy of estimation models that address towards:
(a) Application composition model
(b) Early design stage model
(c) Post architecture stage model
(d) All of the above ( )

28. Data flow diagram is used in:
(a) Architectural design
(b) Interface design
(c) Both a and b
(d) None of the above ( )

29. Entity relationship is used in:
(a) Data design
(b) Architectural design
(c) Interface design
(d) None of the above ( )

30. Knowledge discovery in Databases is also known as:
(a) Data Modeling (b) Data Warehouse
(c) Both a and b (d) Data Structure ( )

31. Cyclomatic complexity provides a :
(a) Quantitative measure (b) Qualitative measure
(c) Both a and b (d) None of the above ( )

32. BVA stands for:
(a) Validation and verification

(b) Variable and values
(c) Verification and validation
(d) None of the above ( )

33. V & V stands for:
(a) Validation and verification
(b) Variable and values
(c) Verification and validation
(d) None of the above ( )

34. Integration testing focus in:
(a) Source Code
(b) Design & Construction of Software Architecture
(c) Both a and b
(d) None of the above ( )

35. Data flow model of an application mainly emphasis on:
(a) Underlying data and the relationship among them
(b) Looping information
(c) Decision and control information
(d) Communication network structure ( )

36. Which is the approach for problem analysis:
(a) Informal analysis
(b) Structured analysis
(c) Object oriented analysis
(d) All of the above ( )

37. In DFD, rectangle is used to represent:
(a) Processes
(b) Flow of data
(c) Source/sink
(d) None of the above ( )

38. A SRS must specify requirement on:
(a) Functionality
(b) External interfaces
(c) Performance
(d) All of the above ( )

39. SRS establishes basis is agreement between the customer and the devloper. The statement is:
(a) True
(b) False
(c) Can't say

(d) None of the above ( )

40. Each phase of spiral model is covers the major activity.
(a) Planning (b) Requirement analysis
(c) Assessment (d) All of the above ( )

Answer Key

1. (b)	2. (d)	3. (c)	4. (c)	5. (c)	6. (b)	7. (a)	8. (d)	9. (c)	10. (a)
11. (b)	12. (a)	13. (a)	14. (a)	15. (d)	16. (b)	17. (a)	18. (b)	19. (b)	20. (a)
21. (c)	22. (d)	23. (b)	24. (d)	25. (b)	26. (a)	27. (d)	28. (a)	29. (b)	30. (b)
31. (a)	32. (b)	33. (a)	34. (b)	35. (d)	36. (a)	37. (b)	38. (d)	39. (c)	40. (c)

____

DESCRIPTIVE PART - II

Year 2008

Time allowed : 2 Hours ***Maximum Marks : 30***

Attempt any four questions out of the six. All questions carry 7½ marks each.

Q.1 Define the term software engineering. Differentiate between program and software. What do you understand by software process?

Q.2 Describe in detail the 4 P's of project management?

Q.3 What is software project planning? Write a note on the approaches suggested by Putnam & Myers to the sizing problem.

Q.4 What are the design principles ? Explain in details the flow of information from analysis model to the design model.

Q.5 What are the objectives of testing? Write down the testing principles write a note on white box testing.

Q.6 What is software architecture ? Why it is important? Define the terms Data Mining and Data Warehousing.

OBJECTIVE PART- I

Year - 2007

Time allowed : One Hour *Maximum Marks : 20*

The question paper contains 40 multiple choice questions with four choices and student will have to pick the correct one. (Each carrying ½ marks.).

1. Pattern recognition and knowledge based system falls under which category of software?
   (a) A.L. (b) Business
   (c) Utility (d) Embedded ( )

2. Which one is not a phase in Water full model?
   (a) Design (b) Code
   (c) Maintenance (d) Recovery ( )

3. Fourth Generation techniques in software engineering involves mainly:
   (a) Advanced design
   (b) Fourth Generation Languages
   (c) Testing tools
   (d) four levels of development ( )

4. Which life model would you use for developing a commercial video game that requires about 8 months of effort from a team of 6 people?
   (a) Opportunistic (b) Waterfall
   (c) Incremental (d) 4 GL ( )

5. Operating systems fall under which category of software?
   (a) Middleware (b) Utility
   (c) Embedded (d) System ( )

6. CORBA falls under which category of software?
   (a) Middleware (b) Utility
   (c) Embedded (d) System ( )

7. Consider the following statements"
   (a) Software development teams need not accept every requirement change proposed by the customer.

(b) Defining requirement basically involves interviewing the customer and documenting what they want
(c) Only A is true
(d) Only is B is true ( )

8. Consider the following statements:
(a) Risk management is about identifying all possible project risk and making sure they are removed
(b) Good quality engineering can prevent defect from happening, not merely find defects and make sure they are fixed
(a) Only (a) is true (b) Only (b) is true
(c) Both (a) and (b) are true (d) Neither (A) nor (B) are true ( )

9. Which is not an element of software matrices domain:
(a) Productivity (b) Quality
(c) Availability (d) Human ( )

10. Which one is not a quality factor (in meaning quality)?
(a) Corrections (b) Usability
(c) Integrity (d) Maintenance ( )

11. Prototyping falls under which phase of software engineering?
(a) Definition (b) Coding
(c) Testing (d) Business ( )

12. ....feasibility involves a study of functions, performance and constrains that may affect the ability to achieve an acceptable system
(a) Economic (b) Technical
(c) Legal (d) Business ( )

13. Architecture Flow Diagrams (AFDs) are drawn in which part of software engineering?
(a) Testing (b) Designing
(c) Specification (d) Maintenance ( )

14. Consider the following statement:
(a) Class diagrams and object diagrams are one and the same
(b) Object diagram reveals instances of various classes and their flows
(a) Only (a) is true
(b) Only (b) is true
(c) Both (a) and (b) are true
(d) Neither (a) nor (b) are true ( )

15. Which one is not an essential part of OOAD?
(a) Class diagram (b) Object diagram

(c) Sequence diagram (d) ER diagram ( )

16. Designing a project requiring FORTRAN language for development would use..........design methodologies.
(a) OOAD (b) Database
(c) Structured (d) Procedural ( )

17. A COM component provides a set of ....for its reusability:
(a) Packages (b) Modules
(c) Interfaces (d) Libraries ( )

18. ....is a collection of class definitions:
(a) Packages (b) Interfaces
(c) Libraries (d) Components ( )

19. A software development using C programming language should incorporate................designing methodologies:
(a) Object oriented (b) Structured
(c) Procedural (d) Database ( )

20. Waterfall model is best suited for software with:
(a) Rigid requirement
(b) Flexible Requirement
(c) Changeable requirement
(d) Requirements are not well understood ( )

21. Which of the following are umbrella activities?
(a) Configuration Control (b) Planning
(c) (a) and (b) both (d) None of the above ( )

22. What is true about ISO 9000 Ceertification and SEI CMM?
(a) SEI CMM is developed for all kinds of roganization while ISO 9000 is specifically developed for software industry
(b) SEI CMM and ISO 9000 was originally developed to assist US department of defense
(c) SEI CMM address the needs or internal improvement of an organization quality while ISO 9000 is a certificate
(d) All of the above ( )

23. Which of the following is nonfunctional requirement?
(a) Execution time (b) Searching capability
(c) Can put data to database (d) Customized user interface ( )

24. Which of the following shows the system behaviour with the external events?
(a) E-R diagrams (b) Data flow diagrams

(c) State transition diagram (d) Data dictionary ( )

25. Consider the following statements:
    (a) It is practically impossible to test every single path through a large module.
    (b) Acceptance tests are usually performed by the development teams.
    (a) Only (a) is true (d) Only (b) is true
    (c) Both (a) and (b) are true (d) Neither (a) nor (b)are true ( )

26. Consider the following statements:
    (a) Is harder to make changes to a design if it has high coupling
    (b) Architectural design is about dividing the system into subsystems with minimal dependencies between them
    (a) Only (a) is true (b) Only (b) is true
    (c) Both (a) and (b) are true (d) Neither (a) nor (b) are true ( )

27. For a good software:
    (a) Coupling should be low, cohesion should be high
    (b) Cohesion should be low, coupling should high
    (c) Both should be low
    (d) Depends on kind of software ( )

28. PERT is:
    (a) Prototyping tool (b) RE-engineering tool
    (c) Change management tool (d) Planning tool ( )

29. In basic COCOMO model effort depends on size, which is in terms of:
    (a) LOC (b) FP
    (c) Time (d) All of the above ( )

30. Consider the following statement:
    (a) If you have an application that has been previously well tested and you add a new feature, you only need to test the new feature, you only need to test the new feature thoroughly, since the other features are known to work.

    (b) If you have access to the source code, it is better to do white box testing. Black box testing is useful primarily when you don't have access to the source code.

    (a) Only (a) is true (b) Only (b) is true
    (c) Both (a and (b) are true (d) Neither (A) nor (b) are true ( )

31. To test the sine function on a calculator, a good combination of input values (in degrees) would be:

(a) –15, 0, 15, 30 45, 60, 75, 90 105
(b) –90, 0 90 180, 270, 360, 450
(c) –55, 0, 30, 90, 135, 630
(d) –30, 0,0 30 90 ( )

32. 'Verification and validation' falls under whose roles:
(a) Designing team (b) Specification team
(c) Documentation team (d) Quality Assurance team ( )

33. A software project inclined to developed a 3D animation game should incorporate which OCOCMO model:
(a) Organic (b) Semidetached
(c) Embedded (d) None of the above ( )

34. Consider the following statement:
(a) Persons (b) Persons/Month
(c) Persons Month (d) Months ( )

35. Consider the following statement:
(a) Reverse engineering deals with getting design from code
(b) Re-engineering deals with getting another product by studying the existing product.
(a) Only (a) is true (b) Only (b) is true
(c) Both (a) and (b) are true (d) Neither (a) nor (b) are true ( )

36. Cyclometric complexity' and 'Basis path' testing falls under which category of testing:"
(a) Black of testing
(b) White box
(c) Both black box and white box
(d) Neither black box nor white box ( )

37. Verification generally deals in testing whether:
(a) The product is according to customer's desire
(b) The product is rightly designed
(c) The product is rightly developed as was designed
(d) The product is useful ( )

38. Unit testing mostly deals in:
(a) Modular testing (b) Procedural testing
(c) Architectural testing (d) None of the above ( )

39. The relative functional strength of a module depends upon:
(a) CISCO (b) Coupling and cohesion
(c) Cohesion (d) Expendability ( )

40. 'CASE' stands for:
    (a) Computer Accessibility and Software Environment
    (b) Computer Aided Software Environment
    (c) Computer Aided Software Engineering
    (d) Computer Aided system Engineering ( )

Answer Key

1. (a)	2. (d)	3. (a)	4. (c)	5. (d)	6. (a)	7. (b)	8. (d)	9. (d)	10. (d)
11. (a)	12. (b)	13. (b)	14. (b)	15. (d)	16. (c)	17. (a)	18. (a)	19. (b)	20. (c)
21. (c)	22. (d)	23. (d)	24. (c)	25. (a)	26. (c)	27. (a)	28. (d)	29. (a)	30. (a)
31. (a)	32. (d)	33. (c)	34. (c)	35. (c)	36. (b)	37. (c)	38. (a)	39. (c)	40. (c)

DESCRIPTIVE PART - II

Year 2007

Time allowed : 2 Hours *Maximum Marks : 30*

Attempt any four questions out of the six. All questions carry 7½ marks each.

Q.1 (a) Explain what wrong the notion that software engineering is too time consuming and interfere with a programmer's productivity.

(b) Compare and contract and spiral model with standard waterfall model, giving specific application areas to illustrate your statements.

Q.2 (a) Explain how size oriented matrices differs from function-oriented matrices, Discuss the pros and Cons of each.

(b) Describe LOC and FP estimation and effort estimation taking an appropriate example.

Q.3 (a) Describe the various mode of COCOMO estimation model giving examples of application falling in each areas.

(b) A project development uses a semi-detached mode of COCOMO model. The effort adjustment factor is taken to be 1.13, estimated KLOC=22. What should be the recommended number of people of the project?

Q.4 For an online banking system draw the Architecture context diagram and 'Architecture flow diagram' describing all its major components.

Q.5 (a) Define and distinguish between the terms 'Verification' and Validation'. What are various activities of SQA team?

(b) Describe Black Box and White Box testing stating various strategies inforporated while performing each.

Q.6 (a) What are Interface ? Describe the OOAD methodologies in short.

(b) Explain the following terms associated with data modeling:

(i) DR diagram

(ii) Data Dictionary

OBJECTIVE PART- I

Year - 2006

Time allowed : One Hour *Maximum Marks : 20*

The question paper contains 40 multiple choice questions with four choices and student will have to pick the correct one. (Each carrying ½ marks.).

1. ....is a collection of programs used to run the system;
   (a) Application software (b) System software
   (c) Testing software (d) None of the above ( )

2. Software engineering is practised through:
   (a) SSAD and OOSAD (b) OOP's and SSAD
   (c) SAD and SSD (d) OOP's and OOSAD ( )

3. Spiral model consists of ....phase
   (a) Two (b) Three
   (c) Four (d) Six ( )

4. The RAD model stands for:
   (a) Rapid Application Development
   (b) Role Action Development
   (c) Rapid and Direct
   (d) Resource Action Development ( )

5. Which one is not manufactured?
   (a) Hardware (b) Software
   (c) Central Processing unit (d) Mouse ( )

6. Which software development model takes more development time?
   (a) Prototype model (b) Waterfall model
   (c) Incremental model (d) Spiral model ( )

7. Which one is not the software development phase?
   (a) Design phase (b) Analysis phase
   (c) Evolution phase (d) Implementation phase ( )

8. FORTRAN means:
   (a) For transaction (b) Formula translation
   (c) Formal translation (d) Foreign translation ( )

9. State which one is not 4 G technique?
   (a) Report generation (b) Data manipulation
   (c) Code generation (D) Programming technique ( )

10. The components of a project include:
   (a) Polities (b) People
   (c) Processes and property (d) All of the above ( )

11. ....is a measure of length of code the software engineering will write to deliver software requirement:
   (a) LOC (b) LCD
   (c) LHC (d) LHC ( )

12. The internal product attribute to describe the software product is:
   (a) Length (b) LCD
   (c) LHC (d) LHD ( )

13. SRM stands for:
   (a) Software Resource Management
   (b) Software Risk Management
   (c) Software Risk Measurement
   (d) Swiss Resources Manager ( )

14. Matrices are...............collection of data, resources and deliverable about project activities:
   (a) Measurement (b) Observation
   (c) Processes (d) None of the above ( )

15. Which one is not the project resources?
   (a) Human resource (b) Observation
   (c) Data management resources (d) None of the above ( )

16. Software matrices are categorized as:
   (a) Product matrices and project matrices
   (b) Project matrices and process matrices
   (c) Process matrices and process matrices
   (d) Project matrices and product matrices ( )

17. CAD/CAE stands for:
   (a) Computer Design Engineering
   (b) Computer Aided Design Engineering
   (c) Computer Aided Design and Computer Aided Engineering
   (d) Computer and Engineering Design ( )

18. The creation and reuse of software building block is:

(a) Usability
(b) Reusability
(c) Adaptability
(d) Non-adaptability ( )

19. Line of code is usually measured in:
(a) Person-month
(b) Per-month
(c) Personal-month
(d) Part-month ( )

20. Which class of software project belongs to COCOMO model?
(a) Organic
(b) Physical
(c) Nonphysical
(d) Inorganic ( )

21. PSPEC means:
(a) Program
(b) Process specification
(c) Personal specification
(d) Procedure specification ( )

22. Which statement is true ?
(a) Abstraction is a means of describing a program function at an appropriate level of details
(b) Abstraction is a means of describing a program structure
(c) Abstraction is a means of describing a program procedure
(d) Abstraction describing a programming technique ( )

23. A module consists of:
(a) Single entry only
(b) Single entry and single exist only
(c) Multiple entries and single exist only
(d) Multiple entries and multiple exists only ( )

24. The interface design of software includes:
(a) Data dictionary
(b) E-R diagram
(c) Data flow diagram
(d) State transition diagram ( )

25. Procedural design and architectural design of software are the components of:
(a) Analysis model
(b) Design model

(c) Data model
(d) Analog design model ( )

26. Flowcharts are drawn:
(a) Before writing the program
(b) After writing the program
(c) After debugging the program
(d) Before testing the program ( )

27. Software design concept is related to:
(a) Algorithms (b) Flowcharts
(c) Modularity (d) Reliability ( )

28. The relative functional strength of a module depends on:
(a) Coupling (b) Coupling and cohesion
(c) Cohesion (d) Expendability ( )

29. Data element definition includes:
(a) Field name (s) (b) Detailed description
(c) Databases (d) Data stores ( )

30. System design goes through............stages.
(a) Logical design (b) Physical design
(c) Logical and physical design (d) Software design ( )

31. The structure chart consists of...................elements.
(a) Two (b) Three
(c) Four (d) Five ( )

32. The interface guidelines on general interaction:
(a) Offer meaningful feedback
(b) Do not allow reversal
(c) NOn-consistency
(d) Allow feedback and control ( )

33. Structured design is .... methodology
(a) Data flow base
(b) Data driven base
(c) Data decomposition base
(d) Data structure base ( )

34. Function overriding of object-oriented allows...................with the same named and parameters in the base class and derived class:
(a) Three Functions
(b) Five Functions

(c) Two Functions
(d) Four Functions ( )

35. State which statement is false:
(a) Testingtime and resources are limited
(b) Use of effective resources to test
(c) Software bugs will always exist in any software module with moderate size
(d) Testing should be performed at the end of module ( )

36. Software testing is performed for:
(a) Verification only
(b) Validation only
(c) Verification and validation only
(d) System designing only ( )

37. System testing does not include:
(a) Recovering testing (b) Alpha testing
(c) Stress testing (d) Security testing ( )

38. Method of equivalence partitioning is:
(a) White box testing
(b) Black box testing
(c) Both (a) and (b)
(d) Grey testing ( )

39. State the correct unit testing step :
(a) Coding and debugging → Integration testing → Unit testing
(b) Coding and debugging → Unit testing → Integration testing
(c) Programming → Codling and debugging → Integration testing
(d) Programming → Debugging → Unit testing ( )

40. State which one is not the characteristic of bugs?
(a) The symptom may be caused by non-errors (e.g. round off, inaccuracies)
(b) The symptom may disappear (temporarily) when another error is corrected
(c) The symptom may be a result of timing problem, rather than processing problems
(d) The symptom may not disappear when another error is corrected ( )

Answer Key

1. (b)	2. (b)	3. (d)	4. (a)	5. (b)	6. (d)	7. (c)	8. (b)	9. (a)	10. (d)
11. (a)	12. (b)	13. (a)	14. (a)	15. (c)	16. (b)	17. (c)	18. (b)	19. (a)	20. (a)
21. (b)	22. (a)	23. (b)	24. (d)	25. (b)	26. (a)	27. (c)	28. (c)	29. (b)	30. (a)
31. (b)	32. (a)	33. (a)	34. (c)	35. (d)	36. (c)	37. (b)	38. (b)	39. (c)	40. (d)

DESCRIPTIVE PART - II

Year 2006

Time allowed : 2 Hours *Maximum Marks : 30*

Attempt any four questions out of the six. All questions carry 7½ marks each.

Q.1 (a) What is Software Engineering ? State the difference between software engineering and traditional engineering.
(b) Differentiate between the characteristics of Hardware and Software.

Q.2 (a) Explain the waterfall model in details.
(b) State the merits and demerits of spiral model.

Q.3 (a) What is COCOMO model?
(b) Consider an office automation system. There are four major modules:
Data entry: 0.6 KLOC
Data update = 0.6 KLOC
Query = 0.8 KLOC
Reports = 1.0 KLOC
The various cost driver attributes are of the high complexity high storage, low experience and low programs capability. Use the intermediate COCOMO model to estimate the final efforts.

Q.4 (a) Describe the following software design concepts:
(i) Abstraction
(ii) Modularity
(b) Explain the difference between:
Architectural design and procedural design.

Q.5 (a) Differentiate between black box and white box testing. Is it correct to claim that if white box testing is done property, it will achieve.

(b) Explain the following OOP's concepts:
(i) Encapsulation
(ii) Inheritance

Q.6 Write detail notes on any one the following:
(a) Project management
(b) Integration testing
(c) Prototype model

Glossary

Abstraction
Abstraction in programming is the process of identifying common patterns that have systematic variations, an abstraction represents the common pattern and provides a means for specifying which variation to use.

Asset
A collection of artefacts.

Aspect migration

The process of *migrating* a software system that is written in a non aspect-oriented way into an aspect-oriented equivalent of that system.

Audit trails
An audit trail establishes additional information about when, why, and by whom changes are made.

Benchmark

(1) A standard against which measurements or comparisons can be made. (2) A recovery file.

Business model

A model of real-world objects and their interactions -or rather, some users' understanding of them

Business rule

A step or set of steps in a process or procedure or guide (algorithmic or heuristic) used by a customer for doing its business, work, or function, and often embodied in whole or in part in the software of a system .

Capability Maturity Model (CMM)

Defined by the Software Engineering Institute (SEI) at Carnegie Mellon University. Describes the level of capability and maturity a software team could aim for and could be assessed against.

Case study

A case study is a research technique where you identify key factors that may affect the outcome of an activity and then document the activity: its inputs, constraints, resources, and outputs.

Cliché

(French) A pattern describing salient features of a concept that supports recognition of that concept in some specified context by application of some specified comparison algorithm.

Clone

Clones are segments of code that are similar according to some definition of similarity.

Commonalities

The set of feature or properties of a component (or system) that are the same, or common, between systems

Decay

Decay is the antithesis of evolution. While the evolution process involves progressive changes, the changes are degenerative in the case of decay.

Domain

A problem area. Typically, many application programs exist to solve the problems in a single domain.

Evolvability

The capability of software products to be evolved to continue to serve its customer in a cost effective way.

Fragile base class problem

Refers to the problem that occurs when independently developed subclasses are broken when their base class evolves.

Framework

A framework is a reusable design of all or part of a software system described by a set of abstract classes and the way instances of those classes collaborate.

Heuristic

Involving or serving as an aid to learning, discovery or problem solving by experimental and especially trial-and-error methods.

Hypothesis

A tentative explanation that accounts for a set of facts and can be tested by further investigation; a theory.

Inconsistency

A state in which two or more overlapping elements of different software models make assertions about aspects of the system they describe which are not jointly satisfiable.

Integrity

The ability of software systems to protect their various components (programs, data) against unauthorized access and modification.

Intercession

Intercession is the ability of a program to modify its own execution state or to alter its own interpretation or meaning.

Introspection

Introspection is the ability of a program to observe and therefore reason about its own state.

Metric

A quantitative measure of the degree to which a system, component or process possesses a given attribute.

Mixin

A mixin is a subclass definition that may be applied to different superclasses to create a related family of modified classes.

Paradigm

A point of view in which some principles, approaches, concepts, and even theories, have been stated uniformly.

Pattern

A standard (object-oriented) design for addressing frequently occuring problems, described in a standard way.

Piecemeal growth

The process of design and implementation in which software is embellished, modified, reduced, enlarged, and improved through a process of repair rather than replacement.

Product line

A collection of existing and potential products that address a coherent business area and share a set of similar characteristics.

Refinement

A refinement is a detailed description that conforms to another (its *abstraction*). Everything said about the abstraction holds, perhaps in a somewhat different form, in the refinement.

Reflection

Reflection is the ability of a program to manipulate as data, something representing the state of the program during its own execution.

Reliability

Software reliability is the probability of a failure-free operation of a computer program in a specified environment for a specified time.

Repairability

Repairability is the ability to facilitate the repair of defects.

Replication

The collection of two or more observations under a set of identical experimental conditions.

Research hypothesis

A tentative theory or supposition provisionally adopted to account for certain facts and to guide in the investigation of others.

Restructuring

A restructuring transformation is often one of appearance, such as altering code to improve its structure in the traditional sense of structured design.

Ripple effect

The phenomenon where a change in one piece of a software system affects at least one other area of the same software system (either directly or indirectly)

Robustness

The ability of software systems to react appropriately to abnormal condition.

Separation of concerns (SOC)

Separation of concerns is closely related to the well-known Roman principle of "divide and conquer".

Software entropy

The amount of disorder in a software system

Software quality assurance

A planned and systematic pattern of all actions necessary to provide adequate confidence that the item or product conforms to established technical requirements.

Traceability

The degree to which a relationship can be established between two or more products of the development process, especially products having a predecessor-successor or master-subordinate relationship to one another.

Treshold

A fixed value (typically an upper bound or lower bound) that distinguishes normal values from abnormal metric values.

Software variability

Refers to the ability of a software sysem or artefact to be efficiently extended, changed, customised or configured for use in a particular context.

Viewpoint

A viewpoint is a specification of the conventions for constructing and using a view.

Wrapper

A software component that encapsulates a system component (a procedure, a program, a file, an API) in order to transform its interface with its environment.

Bibliography

1. Carlo Ghezzi, Mehdi Jazayeri and Dino Mandrioli, Fundamental of Software Engginering
2. Roger S. Pressman, Software Engineering: A Practitioner's Approach, Mcgraw-Hill
3. Ian Sommerville, Software Engineering International Computer Science Series, 8 th Edition
4. Ghezzi, Fundamentals of Software Engineerring 2e