Biyani's Think Tank

Concept based notes

Software Engineering

MCA(IV Sem)

Pallavi Gore
M.Sc.(IT), PGDCA
Lecturer
Deptt. of Information Technology
Biyani Girls College, Jaipur

Published by :
Think Tanks
Biyani Group of Colleges

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

First Edition : 2009

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

Content

□ □ □

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 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:

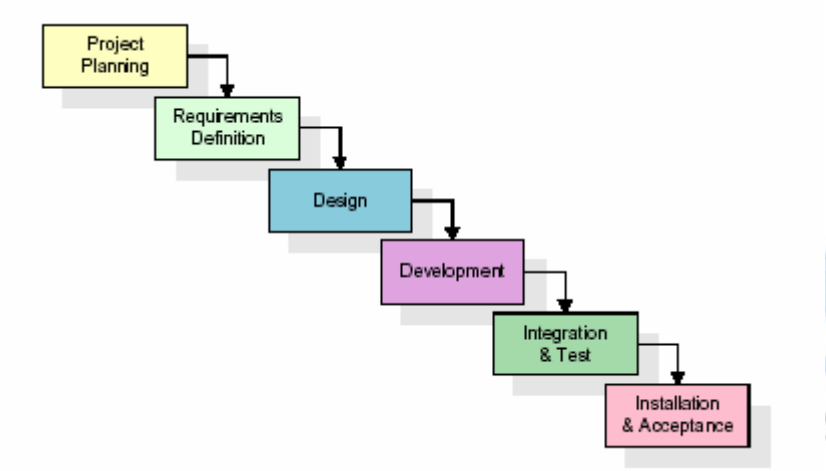


Diagram : Waterfall Model

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.

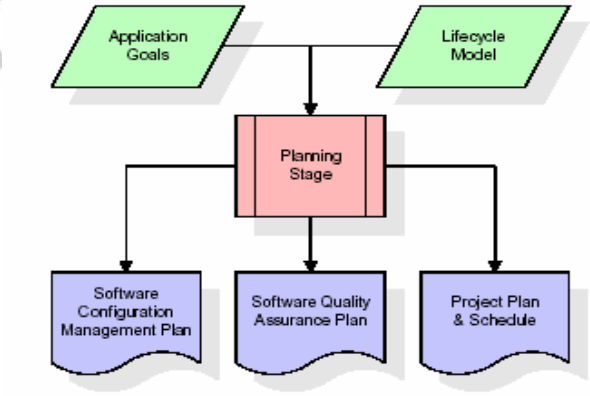


Diagram : Planning

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.

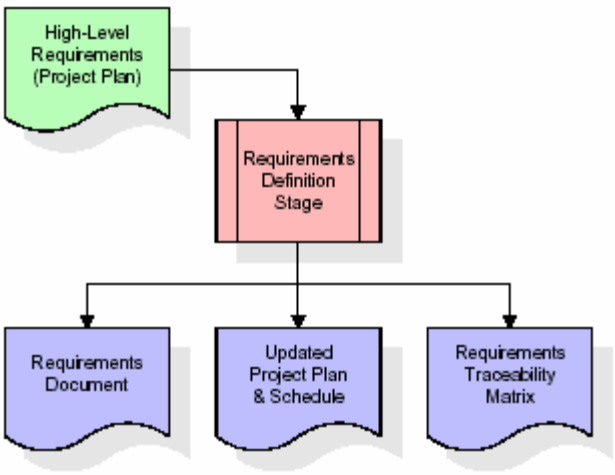


Diagram : Requirement – Definition

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.

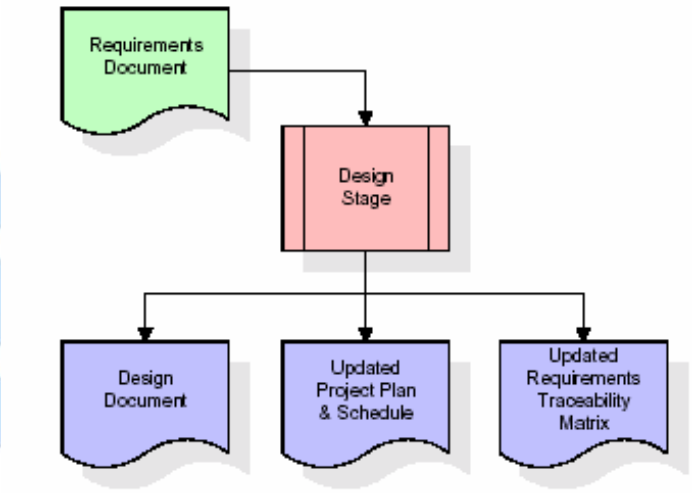


Diagram : Design

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.

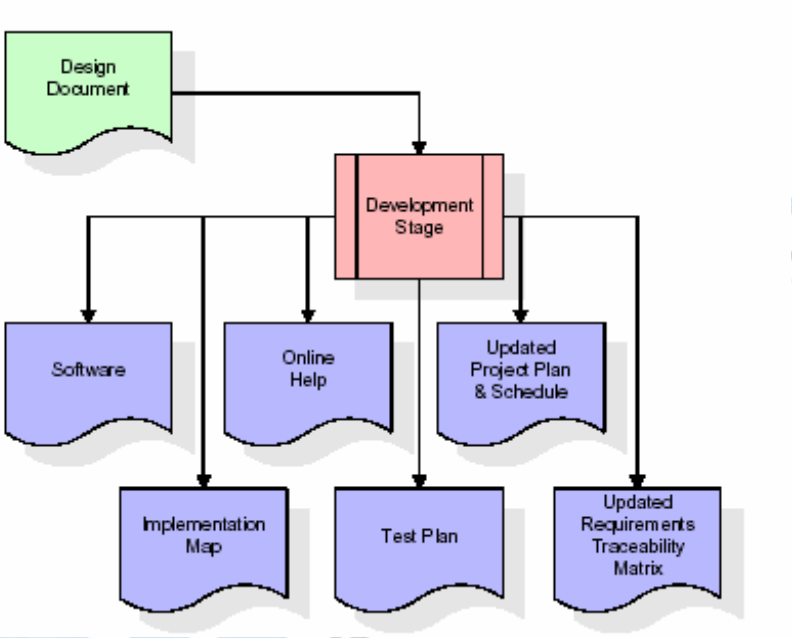


Diagram : Development

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.

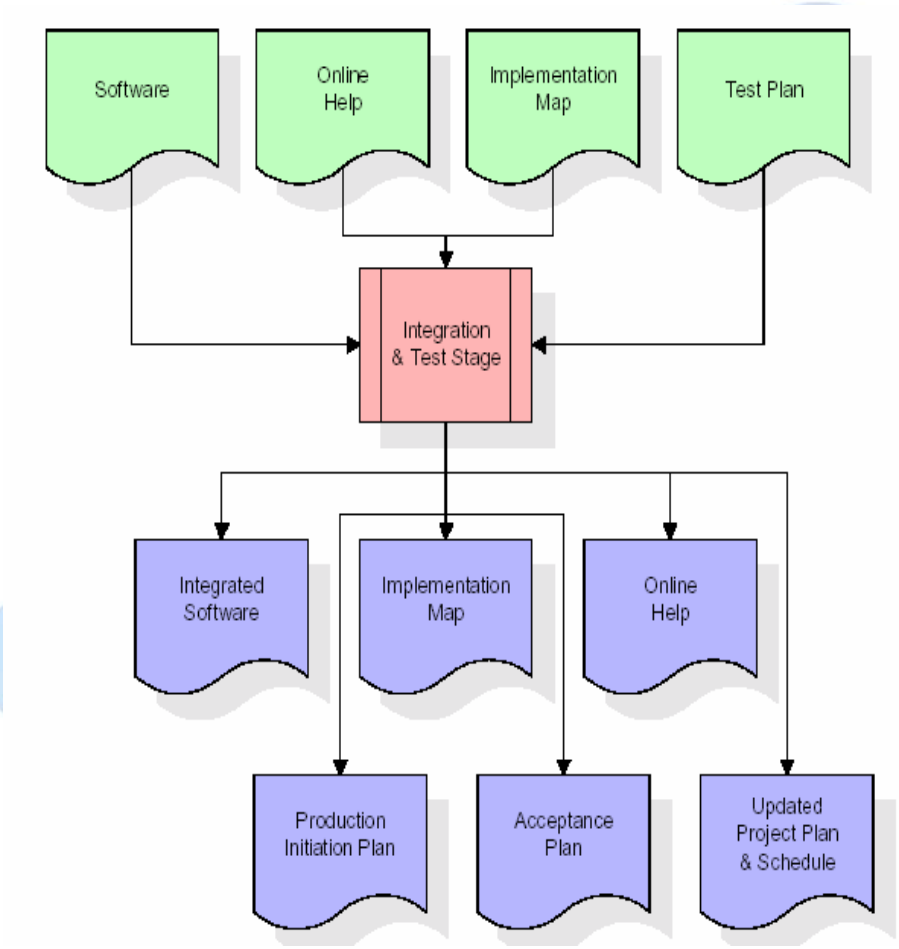


Diagram : Integration & Testing

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.

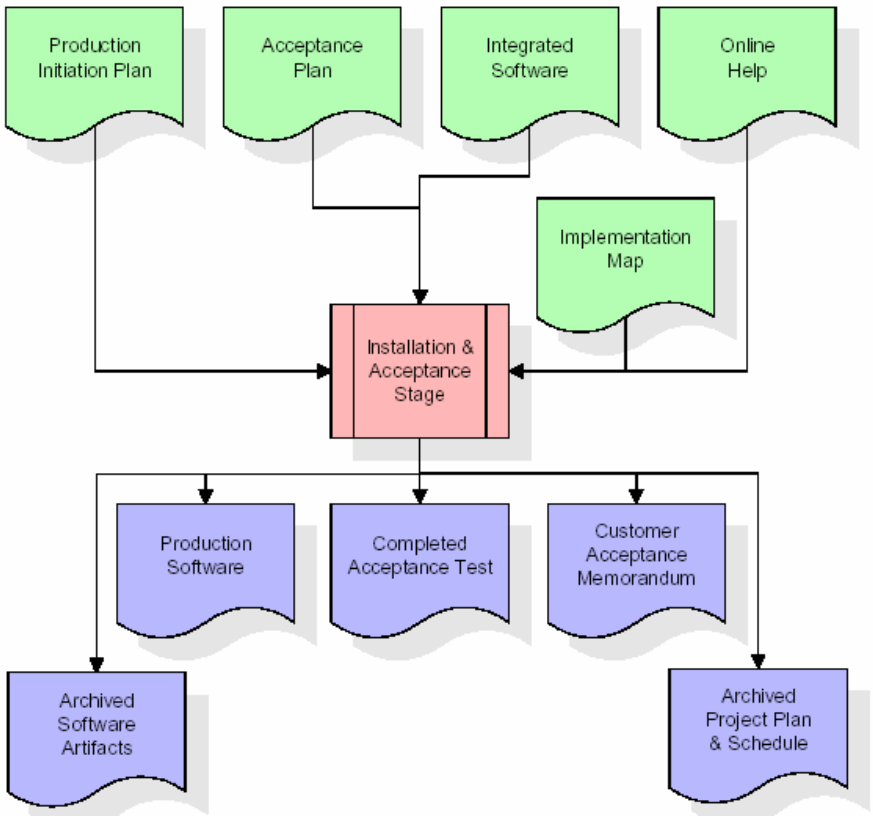


Diagram : Installation & Acceptance

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.4 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.

Q.5 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.6 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

System Analysis

Q.1 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 behaviour "without bothering with the internal points that affect the behaviour.

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.2 List & Describe the different types of Abstraction.

Ans.: Abstractions can be of four types :

i) Data Abstraction

ii) Procedural Abstraction

iii) Control Abstraction

iv) Functionsal 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.3 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.

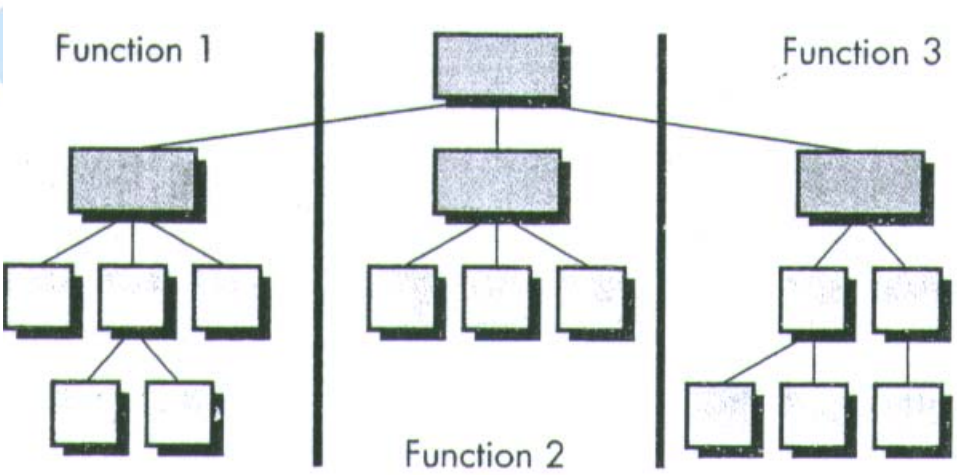
Partitioning could be both vertical as well as horizontal.

Horizontal Partitioning :

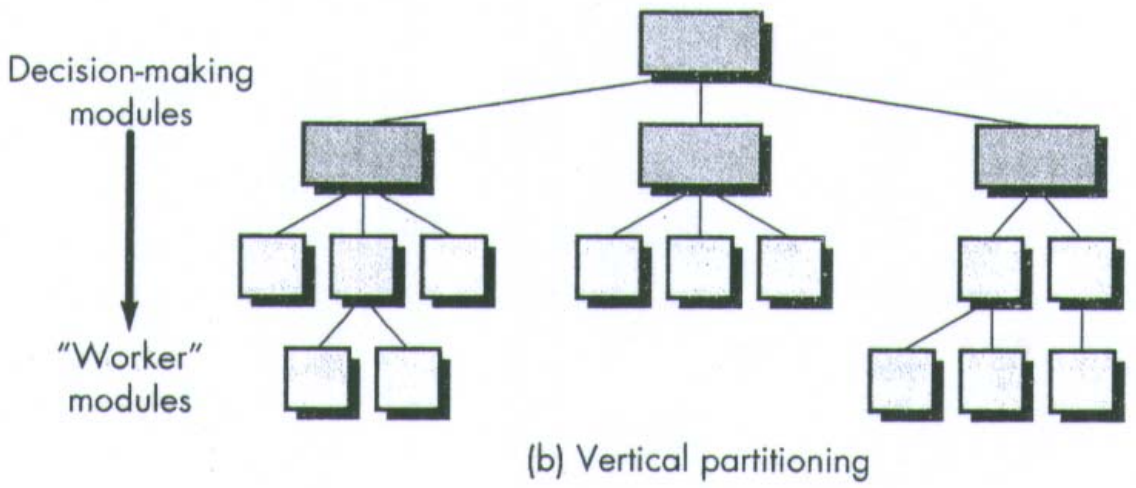
- Horizontal partitioning defines separate branches of the moduler hierarchy.
- Control modules are represented in darker shades.
- Here we do three partitions –
  - (a) Input
  - (b) Data Transformation (Processing)
  - (c) Output
- By the decoupling of major functions changes become less complex.

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 & create 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.

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

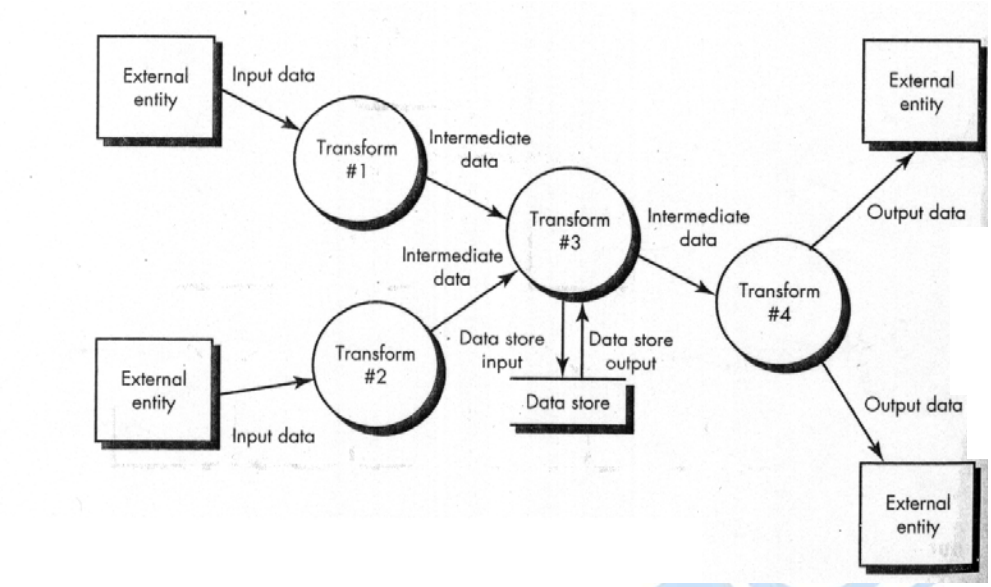


Diagram : Information Flow Model

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, 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 enables 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 uses 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 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.

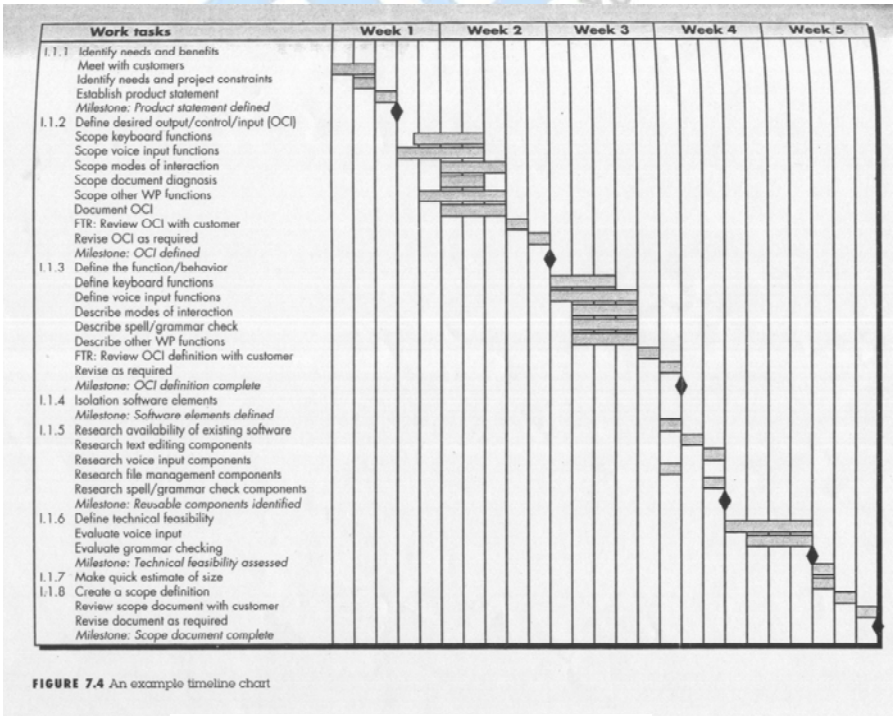


FIGURE 7.4 An example timeline chart

Diagram : Time Line Chart

Q.5 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
<u>Consciousness raising 1</u> 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**
<u>Group management</u> 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.6 What do we mean by project monitoring?

Ans.: It means to keep a careful check of project activities over a period of time.

Q.7 Why should we monitor a project?

Ans.: To work to its full potential, any kind of project needs to set out proposals and objectives. Then a monitoring system should be worked out to keep a check on all the various activities, including finances. This will help project staff to know how things are going, as well as giving early warning of possible problems and difficulties.

Q.8 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.9 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 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 :

i) 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.

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

iii) 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-5

Software Cost & Time Estimation

Q.1 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.2 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.3 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.4 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.5 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.6 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.

(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.7 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 :

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

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

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.8 Explain Reyleigh Distribution Curve.

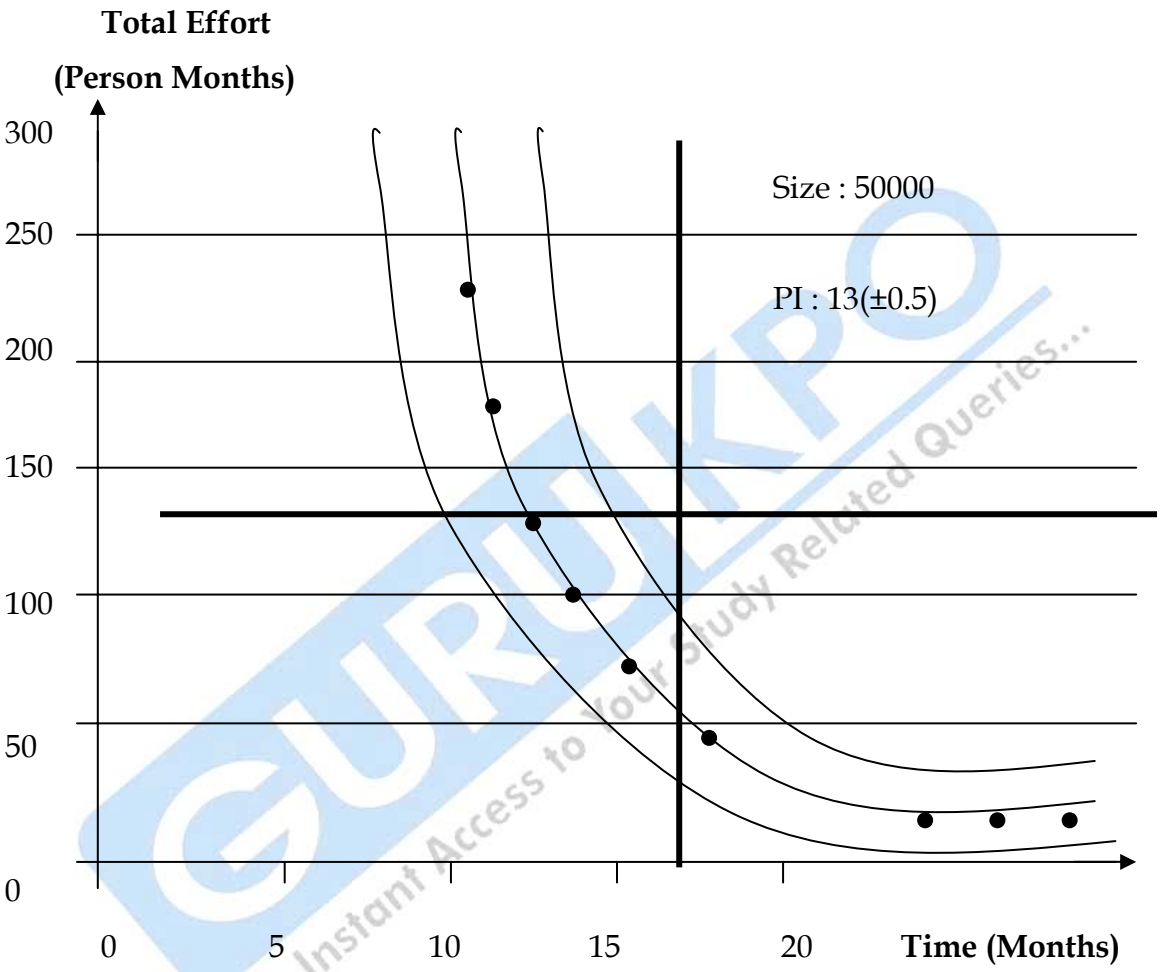
Ans.: The Rayleigh Distribution Curve slopes upward, levels off into a plateau, and then tails off gradually. It is described by the equation :

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

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.



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.9 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.

□ □ □